Pandas: *Pandas is an open-source library that is made mainly for working with relational or labelled data easily.*

- Pandas → Python Data Analysis

To use pandas library, we need to use this library.

Data Structure:

1. DataFrame:

- Two-Dimensional
- Rows and Columns

2. Series:

- 1-Dimensional (like an array)

Use Cases:

1. Reading CSV / Excel / JSON
2. EDA
3. Data Cleaning
4. Encoding (onehot , label)
5. Join table (left, right, inner, outer)
6. Feature Selection

How to create a DataFrame ?

We can create a dataframe by using:

1. numpy array
2. list
3. tuple
4. dict
5. CSV / Excel / JSON

Creation of a dataframe by using numpy array :

In [3]:
```
array = np.random.randint(10,50,size = (6,5))
print(array)
print("-"*80)
df = pd.DataFrame(array,columns = ["A","B","C","D","E"])
df
```

```
[[47 46 15 14 26]
 [31 27 12 42 38]
 [23 20 19 11 23]
 [47 49 14 37 40]
 [41 40 25 21 39]
 [40 43 33 12 35]]
--------------------------------------------------------------------------------
```

Out[3]:

	A	B	C	D	E
0	47	46	15	14	26
1	31	27	12	42	38
2	23	20	19	11	23
3	47	49	14	37	40
4	41	40	25	21	39
5	40	43	33	12	35

DataFrame : df

In [13]:
```
array = np.random.randint(50,100,size = (5,5))
print(array)
print("-"*80)
df = pd.DataFrame(array,columns = list("abcde"))
print("Type of df is:",type(df))        #DataFrame
df
```

```
[[72 63 79 57 93]
 [69 79 79 54 85]
 [77 87 94 53 96]
 [52 56 95 51 75]
 [67 81 79 54 99]]
--------------------------------------------------------------------------------
Type of df is: <class 'pandas.core.frame.DataFrame'>
```

Out[13]:

	a	b	c	d	e
0	72	63	79	57	93
1	69	79	79	54	85
2	77	87	94	53	96
3	52	56	95	51	75
4	67	81	79	54	99

Series : s1, s2

	a	b	c	d	e
0	72	63	79	57	93
1	69	79	79	54	85
2	77	87	94	53	96
3	52	56	95	51	75
4	67	81	79	54	99

```
s1 = df["a"]
print(type(s1))          #series
print(s1)
```

```
<class 'pandas.core.series.Series'>
0    72
1    69
2    77
3    52
4    67
Name: a, dtype: int32
```

```
s2 = df ["c"]
print(type(s2))
s2
```

```
<class 'pandas.core.series.Series'>
```

```
0    79
1    79
2    94
3    95
4    79
Name: c, dtype: int32
```

Labelling of column names and creation of new DF (by adding individual columns) in empty df

```
array = np.random.randint(30,60,size = (5,5))
print(array)
df = pd.DataFrame(array,columns = list("ABCDE"))
df
```

```
[[58 48 46 52 51]
 [56 35 43 41 39]
 [47 42 33 30 42]
 [49 49 38 55 52]
 [51 57 49 45 49]]
```

	A	B	C	D	E
0	58	48	46	52	51
1	56	35	43	41	39
2	47	42	33	30	42
3	49	49	38	55	52
4	51	57	49	45	49

```
df = pd.DataFrame()            #initially empty df
df["A"] = [4,6,8,3,2]
df["B"] = (6,72,7,3,9)
df["C"] = np.array([9,34,78,2,6])
df["D"] = (1,2,5,6,5)
df

```

	A	B	C	D
0	4	6	9	1
1	6	72	34	2
2	8	7	78	5
3	3	3	2	6
4	2	9	6	5

Creation of new DF by using dictionary as input data :

```
data = {"Col_A":np.arange(1,6),
        "Col_B":(6,23,86,24,2)}          #keys >> Column name   and  value >> column values
df = pd.DataFrame(data)
df
```

	Col_A	Col_B
0	1	6
1	2	23
2	3	86
3	4	24
4	5	2

```
df["Col_B"]
```

```
0     6
1    23
2    86
3    24
4     2
Name: Col_B, dtype: int64
```

```
array = np.random.randint(20,80,size = (5,3))
df = pd.DataFrame(array,columns = list("XYZ"))
df
```

	X	Y	Z
0	45	32	56
1	43	50	36
2	43	36	68
3	61	41	55
4	43	49	53

Reading of CSV and Excel files…

```
file_path = r"E:\Python_Learning\Class Notes\JUNE\06_07_Numpy_Pandas\Iris.csv"
file_path
```

```
'E:\\Python_Learning\\Class Notes\\JUNE\\06_07_Numpy_Pandas\\Iris.csv'
```

```
df = pd.read_csv(file_path)
df
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

```
file_path = r"E:\Python_Learning\Class Notes\JUNE\06_07_Numpy_Pandas\Emp_Records.xlsx"
file_path
```

```
'E:\\Python_Learning\\Class Notes\\JUNE\\06_07_Numpy_Pandas\\Emp_Records.xlsx'
```

```
df = pd.read_excel(file_path)
df
```

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509	Lois	36.36	60	13.68	168251	Denver
1	940761	Brenda	47.02	60	9.01	51063	Stonewall
2	428945	Joe	54.15	68	0.98	50155	Michigantown
3	408351	Diane	39.67	51	18.30	180294	Hydetown
4	193819	Benjamin	40.31	58	4.01	117642	Fremont
...	...	...	...	...	...	...	...
95	639892	Jose	22.82	89	1.05	129774	Biloxi
96	704709	Harold	32.61	77	5.93	156194	Carol Stream
97	461593	Nicole	52.66	60	28.53	95673	Detroit
98	392491	Theresa	29.60	57	6.99	51015	Mc Grath
99	495141	Tammy	38.38	55	2.26	93650	Alma

100 rows × 7 columns

```
array1 = np.random.randint(10,50,(4,5))
array1
df = pd.DataFrame(array1,columns =["V","W","X","Y","Z"])
df
```

	V	W	X	Y	Z
0	40	15	16	13	10
1	18	48	29	49	39
2	47	18	40	18	29
3	16	35	14	15	25

```
array1 = np.random.randint(10,50,(4,5))
array1
df = pd.DataFrame(array1,columns =["V","W","X","Y","Z"],index = np.arange(10,14))
df
```

	V	W	X	Y	Z
10	49	19	27	10	22
11	10	22	35	26	22
12	16	49	37	41	45
13	36	32	29	18	44

```
df = pd.DataFrame({"Science":[89,78,88,90,88,85],"English":[88,86,88,90,83,85],"Maths":[98,97,90,82,87,100]},index = list("A
df
```

	Science	English	Maths
A	89	88	98
B	78	86	97
C	88	88	90
D	90	90	82
E	88	83	87
F	85	85	100

```
array2 = np.random.randint(10,70,(5,5))
df = pd.DataFrame(array2,columns = list("ABCDE"), index = list("PQRST"))
df
```

	A	B	C	D	E
P	38	14	26	36	46
Q	27	59	43	12	26
R	14	22	56	34	33
S	24	19	13	11	44
T	47	32	19	60	36

```
file_path = r"E:\Python_Learning\Class Notes\JUNE\06_08_Pandas_Access_n_Missing_Values\movies_data.csv"
file_path
```

```
'E:\\Python_Learning\\Class Notes\\JUNE\\06_08_Pandas_Access_n_Missing_Values\\movies_data.csv'
```

```
movies_df = pd.read_csv(file_path)
movies_df
```

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....
...	...	...	...	...	...	...
974	7.4	Tootsie	PG	Comedy	116	[u'Dustin Hoffman', u'Jessica Lange', u'Teri G...
975	7.4	Back to the Future Part III	PG	Adventure	118	[u'Michael J. Fox', u'Christopher Lloyd', u'Ma...
976	7.4	Master and Commander: The Far Side of the World	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo...
977	7.4	Poltergeist	PG	Horror	114	[u'JoBeth Williams', u"Heather O'Rourke", u'Cr...
978	7.4	Wall Street	R	Crime	126	[u'Charlie Sheen', u'Michael Douglas', u'Tamar...

979 rows × 6 columns

```
movies_df.shape
```

```
(979, 6)
```

```
rows_count = movies_df.shape[0]
columns_count = movies_df.shape[1]
print("Number of rows:",rows_count)
print("Number of columnss:",columns_count)
```

```
Number of rows: 979
Number of columnss: 6
```

```
movies_df.columns                    #column names
```

```
Index(['star_rating', 'title', 'content_rating', 'genre', 'duration',
       'actors_list'],
      dtype='object')
```

```
movies_df.index                  #row names
```

```
RangeIndex(start=0, stop=979, step=1)
```

```
movies_df.dtypes           #datatype in columns
```

```
star_rating       float64
title              object
content_rating     object
genre              object
duration            int64
actors_list        object
dtype: object
```

```
df.info()                  #all information(no. of rows, columns,non-null count,dtype,memory usage)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5 entries, P to T
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       5 non-null      int32
 1   B       5 non-null      int32
 2   C       5 non-null      int32
 3   D       5 non-null      int32
 4   E       5 non-null      int32
dtypes: int32(5)
memory usage: 140.0+ bytes
```

```
print("Number of rows:",titanic_df.shape[0])                    # number of rows
print("Number of columns:",titanic_df.shape[1])                  # number of columns
```

```
Number of rows: 891
Number of columns: 12
```

Null count

```
titanic_df.isna().sum()              # To count the number of null values
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Gender           0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
x = movies_df["star_rating"]            #To acces a particular column
print(x)
type(x)
```

```
0      9.3
1      9.2
2      9.1
3      9.0
4      8.9
      ...
974    7.4
975    7.4
976    7.4
977    7.4
978    7.4
Name: star_rating, Length: 979, dtype: float64
```

```
pandas.core.series.Series
```

```
y = movies_df[["star_rating","genre"]] ##To acces a particular columns,pass column names in a list
print(y)
```

```
     star_rating      genre
0            9.3      Crime
1            9.2      Crime
2            9.1      Crime
3            9.0     Action
4            8.9      Crime
..           ...        ...
974          7.4     Comedy
975          7.4  Adventure
976          7.4     Action
977          7.4     Horror
978          7.4      Crime

[979 rows x 2 columns]
```

df.head

```
First N rows (Default value is 5)
```

```
movies_df.head()                # top five rows >> default value is 5
```

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....

df.tail()

```
Last N rows (Default value is 5)
```

```
movies_df.tail()
```

	star_rating	title	content_rating	genre	duration	actors_list
974	7.4	Tootsie	PG	Comedy	116	[u'Dustin Hoffman', u'Jessica Lange', u'Teri G...
975	7.4	Back to the Future Part III	PG	Adventure	118	[u'Michael J. Fox', u'Christopher Lloyd', u'Ma...
976	7.4	Master and Commander: The Far Side of the World	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo...
977	7.4	Poltergeist	PG	Horror	114	[u'JoBeth Williams', u"Heather O'Rourke", u'Cr...
978	7.4	Wall Street	R	Crime	126	[u'Charlie Sheen', u'Michael Douglas', u'Tamar...

df.iloc

```
df.iloc[row_index, column_index]          #to access specific rows and columns by indexing
```

```
df = pd.DataFrame(np.random.randint(10,50,(5,5)))
print(df)
```

```
    0   1   2   3   4
0  32  15  28  31  42
1  37  30  42  11  36
2  19  23  12  24  37
3  27  35  44  42  24
4  41  43  15  25  36
```

```
df.iloc[2]            # specific row at 2nd index
```

```
0    19
1    23
2    12
3    24
4    37
Name: 2, dtype: int32
```

```
df.iloc[:,3]              # specific column at 3rd index
```

```
0    31
1    11
2    24
3    42
4    25
Name: 3, dtype: int32
```

```
df.iloc[2,2]          # to access particular element
```

```
12
```

```
df.iloc[:,:]
```

	0	1	2	3	4
0	32	15	28	31	42
1	37	30	42	11	36
2	19	23	12	24	37
3	27	35	44	42	24
4	41	43	15	25	36

```
df.iloc[:4,:]
```

	0	1	2	3	4
0	32	15	28	31	42
1	37	30	42	11	36
2	19	23	12	24	37
3	27	35	44	42	24

```
df.iloc[1:3,3]
```

```
1    11
2    24
Name: 3, dtype: int32
```

```
df.iloc[1:3,1:4]
```

	1	2	3
1	30	42	11
2	23	12	24

```
import pandas as pd
import numpy as np
HSC_Result_df = pd.DataFrame({"ENGLISH":[80,73,64],"MARATHI":[86,"NaN",64],"MATHEMATICS & STATICS":[66,"NaN",83],"PHYSICS":[
HSC_Result_df
```

	ENGLISH	MARATHI	MATHEMATICS & STATICS	PHYSICS	CHEMISTRY	BIOLOGY	ANIMAL SCIENCE & DAIRYING	PERCENTAGE
GAYATRI CHAVANKE	80	86	66	62	73	84	NaN	75.17
OM JAGTAP	73	NaN	NaN	61	70	88	193	80.33
AISHWARYA UGALE	64	64	83	85	72	69	NaN	72.83

```
HSC_Result_df.loc[['GAYATRI CHAVANKE',"AISHWARYA UGALE"],["PERCENTAGE"]]
#first add index(row) labels and then column labels
```

	PERCENTAGE
GAYATRI CHAVANKE	75.17
AISHWARYA UGALE	72.83

```
HSC_Result_df.iloc[[0,2],[7]]
```

	PERCENTAGE
GAYATRI CHAVANKE	75.17
AISHWARYA UGALE	72.83

```
movies_df.iloc[[0,1,2,3,4,5],[0,1,4]]    #particular rows (without slicing) >> use square brackets
```

	star_rating	title	duration
0	9.3	The Shawshank Redemption	142
1	9.2	The Godfather	175
2	9.1	The Godfather: Part II	200
3	9.0	The Dark Knight	152
4	8.9	Pulp Fiction	154
5	8.9	12 Angry Men	96

```
movies_df.iloc[1:5,[0,1,4]]          # for range of rows or columns (slicing) >> put directly values without sq. brackets
```

	star_rating	title	duration
1	9.2	The Godfather	175
2	9.1	The Godfather: Part II	200
3	9.0	The Dark Knight	152
4	8.9	Pulp Fiction	154

```
df.loc[row_labels, column_lables]
```

```
movies_df.head()
```

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....

```
movies_df.loc[1:3]
```

	star_rating	title	content_rating	genre	duration	actors_list
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...

```
movies_df.loc[1:3,"title"]       #series
```

```
1            The Godfather
2    The Godfather: Part II
3           The Dark Knight
Name: title, dtype: object
```

```
1 movies_df.loc[[10,20,30,40,50,100],["star_rating","title"]] #df
```

	star_rating	title
10	8.8	The Lord of the Rings: The Fellowship of the Ring
20	8.7	The Matrix
30	8.6	Spirited Away
40	8.5	The Green Mile
50	8.5	Cinema Paradiso
100	8.3	The Treasure of the Sierra Madre

```
1 movies_df.loc[10:200,["star_rating","title","duration"]]
```

	star_rating	title	duration
10	8.8	The Lord of the Rings: The Fellowship of the Ring	178
11	8.8	Inception	148
12	8.8	Star Wars: Episode V - The Empire Strikes Back	124
13	8.8	Forrest Gump	142
14	8.8	The Lord of the Rings: The Two Towers	179
...	...	...	...
196	8.1	Guardians of the Galaxy	121
197	8.1	In the Name of the Father	133
198	8.1	Kill Bill: Vol. 1	111
199	8.1	No Country for Old Men	122
200	8.1	Le Samourai	101

191 rows × 3 columns

```
df = pd.DataFrame(np.random.randint(10,100,(4,5)),index = list("abcd"),columns = ["P","Q","R","S","T"])
df
```

	P	Q	R	S	T
a	35	53	29	43	44
b	19	78	78	44	32
c	92	37	20	25	14
d	32	39	60	34	80

```
df.iloc[1:4,1:5]
```

	Q	R	S	T
b	78	78	44	32
c	37	20	25	14
d	39	60	34	80

```
df.loc[["a","d"],["Q","T"]]
```

	Q	T
a	53	44
d	39	80

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randint(10,100,(4,5)),index = list("abcd"),columns = ["P","Q","R","S","T"])
df.loc["a":"d",["Q","T"]]
```

	Q	T
a	50	87
b	16	30
c	65	41
d	70	90

```
np.nan  >> Null Values
NaN in dataframe
```

Check null values

```
1 titanic_df.isna()
```

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...
886	False	False	False	False	False	False	False	False	False	False	True	False
887	False	False	False	False	False	False	False	False	False	False	False	False
888	False	False	False	False	False	True	False	False	False	False	True	False
889	False	False	False	False	False	False	False	False	False	False	False	False
890	False	False	False	False	False	False	False	False	False	False	True	False

891 rows × 12 columns

```
1 titanic_df.isnull()
```

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...
886	False	False	False	False	False	False	False	False	False	False	True	False
887	False	False	False	False	False	False	False	False	False	False	False	False
888	False	False	False	False	False	True	False	False	False	False	True	False

```
1 titanic_df.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Gender           0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
1 titanic_df.isna().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Gender           0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
1 titanic_df.isna().mean()*100
```

```
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Gender          0.000000
Age            19.865320
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.000000
Cabin          77.104377
Embarked        0.224467
dtype: float64
```

```
df.fillna()              #df
df[col_name].fillna()       #series
```

```
emp_df.fillna(40).head(10)        # To fill the value in all columns
```

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509	Lois	36.36	60.0	13.68	40.0	Denver
1	940761	Brenda	40.00	40.0	9.01	51063.0	Stonewall
2	428945	Joe	40.00	68.0	40.00	50155.0	Michigantown
3	408351	Diane	40.00	40.0	18.30	40.0	Hydetown
4	193819	Benjamin	40.31	58.0	4.01	117642.0	Fremont
5	499687	Patrick	34.86	58.0	12.02	40.0	Macksburg
6	539712	Nancy	22.14	50.0	0.87	98189.0	Atlanta
7	380086	Carol	59.12	40.0	34.52	60918.0	Blanchester
8	477616	Frances	58.18	42.0	23.27	121587.0	Delmita
9	162402	Diana	29.73	60.0	3.44	43010.0	Eureka Springs

```
emp_df["Age in Yrs"] = emp_df["Age in Yrs"].fillna(40)
emp_df                                    # To fill the values at specific column
```

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509	Lois	36.36	60.0	13.68	NaN	Denver
1	940761	Brenda	40.00	NaN	9.01	51063.0	Stonewall
2	428945	Joe	40.00	68.0	NaN	50155.0	Michigantown
3	408351	Diane	40.00	NaN	18.30	NaN	Hydetown
4	193819	Benjamin	40.31	58.0	4.01	117642.0	Fremont
...	...	...	...	...	...	...	...
95	639892	Jose	22.82	89.0	1.05	129774.0	Biloxi

```
emp_df["Weight in Kgs"].fillna(50,inplace = True)
emp_df
```

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509	Lois	36.36	60.0	13.68	NaN	Denver
1	940761	Brenda	40.00	50.0	9.01	51063.0	Stonewall
2	428945	Joe	40.00	68.0	NaN	50155.0	Michigantown
3	408351	Diane	40.00	50.0	18.30	NaN	Hydetown
4	193819	Benjamin	40.31	58.0	4.01	117642.0	Fremont
...	...	...	...	...	...	...	...
95	639892	Jose	22.82	89.0	1.05	129774.0	Biloxi
96	704709	Harold	32.61	77.0	5.93	156194.0	Carol Stream
97	461593	Nicole	52.66	60.0	28.53	95673.0	Detroit
98	392491	Theresa	29.60	57.0	6.99	51015.0	Mc Grath
99	495141	Tammy	38.38	55.0	2.26	93650.0	Alma

100 rows × 7 columns

```
emp_df.isna().sum()
```

```
Emp ID           0
First Name       0
Age in Yrs       0
Weight in Kgs    0
Age in Company   1
Salary           3
City             0
dtype: int64
```

.describe()

```
1 emp_df.describe()
```

	Emp ID	Age in Yrs	Weight in Kgs	Age in Company	Salary
count	100.00000	97.000000	98.000000	99.000000	97.000000
mean	547652.10000	39.000309	58.132653	9.059192	119102.670103
std	257664.16679	12.130049	12.397737	8.663444	45969.343158
min	134841.00000	21.100000	40.000000	0.020000	42005.000000
25%	328643.75000	28.020000	50.000000	2.275000	84318.000000
50%	497414.00000	37.270000	56.000000	6.940000	117642.000000
75%	766040.00000	49.850000	61.750000	13.845000	160623.000000
max	979607.00000	59.470000	90.000000	34.520000	197537.000000

```
1 emp_df["Age in Yrs"].mean()
```

```
39.00030927835053
```

```
1 emp_df["Weight in Kgs"].median()
```

```
56.0
```

```
1 mean = emp_df["Weight in Kgs"].mean()
2 mean
```

```
58.13265306122449
```

```
1 emp_df["Weight in Kgs"].mode()[0]
```

```
60.0
```

```
emp_df["Weight in Kgs"].fillna(emp_df["Weight in Kgs"].mean(),inplace =True)
emp_df
```

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509	Lois	36.36	60.000000	13.68	NaN	Denver
1	940761	Brenda	NaN	58.132653	9.01	51063.0	Stonewall
2	428945	Joe	NaN	68.000000	NaN	50155.0	Michigantown
3	408351	Diane	NaN	58.132653	18.30	NaN	Hydetown
4	193819	Benjamin	40.31	58.000000	4.01	117642.0	Fremont
...	...	...	...	...	...	...	...
95	639892	Jose	22.82	89.000000	1.05	129774.0	Biloxi
96	704709	Harold	32.61	77.000000	5.93	156194.0	Carol Stream
97	461593	Nicole	52.66	60.000000	28.53	95673.0	Detroit
98	392491	Theresa	29.60	57.000000	6.99	51015.0	Mc Grath
99	495141	Tammy	38.38	55.000000	2.26	93650.0	Alma

100 rows × 7 columns

```
emp_df.iloc[0,4] =  np.nan
emp_df
```

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509	Lois	36.36	60.000000	NaN	NaN	Denver
1	940761	Brenda	NaN	58.132653	9.01	51063.0	Stonewall
2	428945	Joe	NaN	68.000000	NaN	50155.0	Michigantown
3	408351	Diane	NaN	58.132653	18.30	NaN	Hydetown
4	193819	Benjamin	40.31	58.000000	4.01	117642.0	Fremont

```
titanic_df.dropna()        # by default axis = 0  i.e row will be dropped
                           #Row having even a single null value, will be deleted...
                          # after applying dropna function,there are 183 rows only(before applying function - 891 rows)
                            #and all these 183 rows will be non-null
```

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S
...	...	...	...	...	...	...	...	...	...	...	...	...
871	872	1	1	Beckwith, Mrs. Richard Leonard (Sallie Monypeny)	female	47.0	1	1	11751	52.5542	D35	S
872	873	0	1	Carlsson, Mr. Frans Olof	male	33.0	0	0	695	5.0000	B51 B53 B55	S
879	880	1	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11767	83.1583	C50	C
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C

183 rows × 12 columns

```
titanic_df.dropna(axis = 1) # if axis = 1  i.e column will be dropped
                            # Column having even a single null value, will be deleted...
                  # after applying dropna function,there are 9 columns only(before applying dropna function,columns-12)
                              #and all these 9 columns will be non-null
```

	PassengerId	Survived	Pclass	Name	Gender	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	0	0	373450	8.0500
...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	0	0	211536	13.0000
887	888	1	1	Graham, Miss. Margaret Edith	female	0	0	112053	30.0000
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	1	2	W./C. 6607	23.4500
889	890	1	1	Behr, Mr. Karl Howell	male	0	0	111369	30.0000
890	891	0	3	Dooley, Mr. Patrick	male	0	0	370376	7.7500

891 rows × 9 columns

Drop columns: (axis = 1)

Unnecessary columns : Columns having unique values such as Name, Passenger ID, Roll numbers, ticket numbers, serial numbers

These columns will never contribute in your model. It will just reduce the accuracy because the models will get trained automatically if present in dataset. So, for better accuracy it will be better if dropped such unnecessary columns

If **inplace = True** ,main dataframe will be updated

```
titanic_df.drop(["PassengerId","Name","Ticket","Cabin"],axis = 1,inplace = True)
# To drop multiple columns ,pass a list of all those column names.
```

	Survived	Pclass	Gender	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S
...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S
887	1	1	female	19.0	0	0	30.0000	S
888	0	3	female	NaN	1	2	23.4500	S
889	1	1	male	26.0	0	0	30.0000	C
890	0	3	male	32.0	0	0	7.7500	Q

891 rows × 8 columns

```
emp_df.dropna() # rows having even a single null value,will be dropped
```

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
4	193819.0	Benjamin	40.31	58.0	4.01	117642.0	Fremont
6	539712.0	Nancy	22.14	50.0	0.87	98189.0	Atlanta
8	477616.0	Frances	58.18	42.0	23.27	121587.0	Delmita
10	231469.0	Ralph	42.50	80.0	8.29	118457.0	Sabetha
13	301576.0	Wayne	21.10	87.0	0.02	92758.0	Maida
...	...	...	...	...	...	...	...
95	639892.0	Jose	22.82	89.0	1.05	129774.0	Biloxi
96	704709.0	Harold	32.61	77.0	5.93	156194.0	Carol Stream
97	461593.0	Nicole	52.66	60.0	28.53	95673.0	Detroit
98	392491.0	Theresa	29.60	57.0	6.99	51015.0	Mc Grath
99	495141.0	Tammy	38.38	55.0	2.26	93650.0	Alma

91 rows × 7 columns

Threshold

```
emp_df.dropna(thresh = 7) # we will get 7 non-null columns. here default is 7 >> emp_df.shape[1]
```

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
4	193819.0	Benjamin	40.31	58.0	4.01	117642.0	Fremont
6	539712.0	Nancy	22.14	50.0	0.87	98189.0	Atlanta
8	477616.0	Frances	58.18	42.0	23.27	121587.0	Delmita
10	231469.0	Ralph	42.50	80.0	8.29	118457.0	Sabetha
13	301576.0	Wayne	21.10	87.0	0.02	92758.0	Maida
...	...	...	...	...	...	...	...
95	639892.0	Jose	22.82	89.0	1.05	129774.0	Biloxi
96	704709.0	Harold	32.61	77.0	5.93	156194.0	Carol Stream
97	461593.0	Nicole	52.66	60.0	28.53	95673.0	Detroit
98	392491.0	Theresa	29.60	57.0	6.99	51015.0	Mc Grath
99	495141.0	Tammy	38.38	55.0	2.26	93650.0	Alma

91 rows × 7 columns

```
emp_df.dropna(thresh = 6) # 6 non-null(i.e 1 null) values in a row will be accepted.
                          # rows having more than 1 null values will be deleted
```

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509.0	Lois	36.36	60.0	13.68	NaN	Denver
4	193819.0	Benjamin	40.31	58.0	4.01	117642.0	Fremont
6	539712.0	Nancy	22.14	50.0	0.87	98189.0	Atlanta
8	477616.0	Frances	58.18	42.0	23.27	121587.0	Delmita
9	162402.0	Diana	29.73	NaN	3.44	43010.0	Eureka Springs
...	...	...	...	...	...	...	...
95	639892.0	Jose	22.82	89.0	1.05	129774.0	Biloxi
96	704709.0	Harold	32.61	77.0	5.93	156194.0	Carol Stream
97	461593.0	Nicole	52.66	60.0	28.53	95673.0	Detroit
98	392491.0	Theresa	29.60	57.0	6.99	51015.0	Mc Grath
99	495141.0	Tammy	38.38	55.0	2.26	93650.0	Alma

95 rows × 7 columns

```
emp_df.dropna(thresh = 5) # 5 non-null(i.e 2 null) values in a row will be accepted
                          # rows having more than 2 null values will be deleted
                        # we may conclude,if threshold value decreases, number of rows may increase
```

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509.0	Lois	36.36	60.0	13.68	NaN	Denver
1	940761.0	Brenda	NaN	NaN	9.01	51063.0	Stonewall
2	428945.0	Joe	NaN	68.0	NaN	50155.0	Michigantown
4	193819.0	Benjamin	40.31	58.0	4.01	117642.0	Fremont
5	499687.0	Patrick	34.86	NaN	12.02	NaN	Macksburg
...	...	...	...	...	...	...	...
95	639892.0	Jose	22.82	89.0	1.05	129774.0	Biloxi
96	704709.0	Harold	32.61	77.0	5.93	156194.0	Carol Stream
97	461593.0	Nicole	52.66	60.0	28.53	95673.0	Detroit
98	392491.0	Theresa	29.60	57.0	6.99	51015.0	Mc Grath
99	495141.0	Tammy	38.38	55.0	2.26	93650.0	Alma

99 rows × 7 columns

```
emp_df.dropna(thresh = 3,axis = 0) # 3 non-null(i.e 4 null) values in a row will be accepted
                          # rows having more than 4 null values will be deleted
                        # axis = 0 is default
```

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509.0	Lois	36.36	60.0	13.68	NaN	Denver
1	940761.0	Brenda	NaN	NaN	9.01	51063.0	Stonewall
2	428945.0	Joe	NaN	68.0	NaN	50155.0	Michigantown
3	NaN	Diane	NaN	NaN	18.30	NaN	Hydetown

```
emp_df.dropna(thresh = 14,axis = 1) # 14 non-null(i.e 1 null) values in a column will be accepted
                                    # columns having more than 1 null values will be deleted
```

	Emp ID	First Name	City
0	677509	Lois	Denver
1	940761	Brenda	Stonewall
2	428945	Joe	Michigantown
3	438658	Diane	Hydetown
4	193819	Benjamin	Fremont
5	499687	Patrick	Macksburg
6	539712	Nancy	Atlanta
7	380086	Carol	NaN
8	477616	Frances	Delmita
9	162402	Diana	Eureka Springs
10	231469	Ralph	Sabetha
11	153989	Jack	Las Vegas
12	386158	Melissa	New Matamoras
13	301576	Wayne	Maida
14	441771	Cheryl	Quecreek

```
emp_df.isna().sum()    #Emp ID and First Name column is having 0 null values
```

```
Emp ID            0
First Name        0
Age in Yrs        5
Weight in Kgs     4
Age in Company    2
Salary            3
City              1
dtype: int64
```

```
emp_df.dropna(axis = 1)  # column having 0 null values will be printed
                         # Here by default, threshold  = emp_df.shape[0]
```

	Emp ID	First Name
0	677509	Lois
1	940761	Brenda
2	428945	Joe
3	438658	Diane
4	193819	Benjamin
5	499687	Patrick
6	539712	Nancy
7	380086	Carol
8	477616	Frances
9	162402	Diana
10	231469	Ralph
11	153989	Jack
12	386158	Melissa
13	301576	Wayne

```
emp_df.dropna(thresh = 12,axis = 1) # 12 non-null(i.e 3 null) values in a column will be accepted
                                    # columns having more than 3 null values will be deleted
```

	Emp ID	First Name	Age in Company	Salary	City
0	677509	Lois	13.68	NaN	Denver
1	940761	Brenda	9.01	51063.0	Stonewall
2	428945	Joe	NaN	50155.0	Michigantown
3	438658	Diane	18.30	NaN	Hydetown
4	193819	Benjamin	4.01	117642.0	Fremont
5	499687	Patrick	12.02	NaN	Macksburg
6	539712	Nancy	0.87	98189.0	Atlanta
7	380086	Carol	34.52	60918.0	NaN
8	477616	Frances	23.27	121587.0	Delmita
9	162402	Diana	3.44	43010.0	Eureka Springs
10	231469	Ralph	8.29	118457.0	Sabetha
11	153989	Jack	NaN	82965.0	Las Vegas
12	386158	Melissa	1.68	166892.0	New Matamoras
13	301576	Wayne	0.02	92758.0	Maida
14	441771	Cheryl	26.69	92220.0	Quecreek

Sorting Functions:

1) sort_index()
2) sort_values()
3) set_index()
4) reset_index()

```
1 movies_df.sort_index(axis = 0)  # used to sort row index
```

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....
...	...	...	...	...	...	...
974	7.4	Tootsie	PG	Comedy	116	[u'Dustin Hoffman', u'Jessica Lange', u'Teri G...
975	7.4	Back to the Future Part III	PG	Adventure	118	[u'Michael J. Fox', u'Christopher Lloyd', u'Ma...
976	7.4	Master and Commander: The Far Side of the World	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo...
977	7.4	Poltergeist	PG	Horror	114	[u'JoBeth Williams', u"Heather O'Rourke", u'Cr...
978	7.4	Wall Street	R	Crime	126	[u'Charlie Sheen', u'Michael Douglas', u'Tamar...

979 rows × 6 columns

```
1 movies_df.sort_index(axis = 1)  # used to sort column names
```

	actors_list	content_rating	duration	genre	star_rating	title
0	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...	R	142	Crime	9.3	The Shawshank Redemption
1	[u'Marlon Brando', u'Al Pacino', u'James Caan']	R	175	Crime	9.2	The Godfather
2	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...	R	200	Crime	9.1	The Godfather: Part II
3	[u'Christian Bale', u'Heath Ledger', u'Aaron E...	PG-13	152	Action	9.0	The Dark Knight

```
import numpy as np
import pandas as pd
a_df = pd.DataFrame(np.random.randint(10,30,size = (6,4)),index = list('QWERTY'),columns = list('pyth'))
a_df
```

	p	y	t	h
Q	11	25	14	28
W	24	10	19	18
E	22	21	26	15
R	16	20	14	12
T	23	19	22	26
Y	26	10	25	12

```
a_df.sort_index()  # sort rows by row index  #default axis = 0
```

	p	y	t	h
E	22	21	26	15
Q	11	25	14	28
R	16	20	14	12
T	23	19	22	26
W	24	10	19	18
Y	26	10	25	12

```
a_df.sort_index(axis = 1)   # sort columns by column names
```

	h	p	t	y
Q	28	11	14	25
W	18	24	19	10
E	15	22	26	21
R	12	16	14	20
T	26	23	22	19
Y	12	26	25	10

```
a_df.sort_values("h")
```

	p	y	t	h
R	16	20	14	12
Y	26	10	25	12
E	22	21	26	15
W	24	10	19	18
T	23	19	22	26
Q	11	25	14	28

```
movies_df.sort_values("duration")
```

	star_rating	title	content_rating	genre	duration	actors_list
389	8.0	Freaks	UNRATED	Drama	64	[u'Wallace Ford', u'Leila Hyams', u'Olga Bacla...
338	8.0	Battleship Potemkin	UNRATED	History	66	[u'Aleksandr Antonov', u'Vladimir Barsky', u'G...
258	8.1	The Cabinet of Dr. Caligari	UNRATED	Crime	67	[u'Werner Krauss', u'Conrad Veidt', u'Friedric...
293	8.1	Duck Soup	PASSED	Comedy	68	[u'Groucho Marx', u'Harpo Marx', u'Chico Marx']
88	8.4	The Kid	NOT RATED	Comedy	68	[u'Charles Chaplin', u'Edna Purviance', u'Jack...
...	...	...	...	...	...	...
445	7.9	The Ten Commandments	APPROVED	Adventure	220	[u'Charlton Heston', u'Yul Brynner', u'Anne Ba...
142	8.3	Lagaan: Once Upon a Time in India	PG	Adventure	224	[u'Aamir Khan', u'Gracy Singh', u'Rachel Shell...
78	8.4	Once Upon a Time in America	R	Crime	229	[u'Robert De Niro', u'James Woods', u'Elizabet...
157	8.2	Gone with the Wind	G	Drama	238	[u'Clark Gable', u'Vivien Leigh', u'Thomas Mit...
476	7.8	Hamlet	PG-13	Drama	242	[u'Kenneth Branagh', u'Julie Christie', u'Dere...

```
a_df.sort_values("h")
```

	p	y	t	h
R	16	20	14	12
Y	26	10	25	12
E	22	21	26	15
W	24	10	19	18
T	23	19	22	26
Q	11	25	14	28

```
a_df.sort_values(["h","y"])  # if there are two same values in a sorted column,then sort by another column
   #for that similar row values(12(R),12(Y)) of initially sorted column(here,"h"),if we sorted next
    #by "y" column then sorting will be according to those two rows only for that column"y"
                # sort_values in DataScience is mostly(99%) by columns #so, no need to give axis input
```

	p	y	t	h
Y	26	10	25	12
R	16	20	14	12
E	22	21	26	15
W	24	10	19	18
T	23	19	22	26
Q	11	25	14	28

To access and update value...

```
b_df = pd.DataFrame(np.random.randint(10,100,(5,5)),index = list("ABCDE"),columns = ["P",'Q','R','S','T'])
b_df
```

	P	Q	R	S	T
A	93	41	11	34	36
B	35	83	61	66	33
C	44	70	34	87	70
D	58	26	74	40	38
E	56	85	45	70	26

```
# To update value in "T" column and "C" row by 100, use iloc
b_df.iloc[2,4] = 100
b_df
```

	P	Q	R	S	T
A	93	41	11	34	36
B	35	83	61	66	33
C	44	70	34	87	100
D	58	26	74	40	38
E	56	85	45	70	26

```
movies_df.sort_values("star_rating",ascending = True)    #star_rating column will be sorted
                    #during sorting data will remain same for row and column,only we sort it by our choice
                    #ascending = True is default here
```

	star_rating	title	content_rating	genre	duration	actors_list
978	7.4	Wall Street	R	Crime	126	[u'Charlie Sheen', u'Michael Douglas', u'Tamar...
950	7.4	Bound	R	Crime	108	[u'Jennifer Tilly', u'Gina Gershon', u'Joe Pan...
949	7.4	Home Alone	PG	Comedy	103	[u'Macaulay Culkin', u'Joe Pesci', u'Daniel St...
948	7.4	Frances Ha	R	Comedy	86	[u'Greta Gerwig', u'Mickey Sumner', u'Adam Dri...
947	7.4	Eraserhead	UNRATED	Drama	89	[u'Jack Nance', u'Charlotte Stewart', u'Allen ...
...	...	...	...	...	...	...
6	8.9	The Good, the Bad and the Ugly	NOT RATED	Western	161	[u'Clint Eastwood', u'Eli Wallach', u'Lee Van ...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...

979 rows × 6 columns

```
movies_df.sort_values(["genre","star_rating"],ascending = False)
                    # sorting will be in descending order
```

	star_rating	title	content_rating	genre	duration	actors_list
6	8.9	The Good, the Bad and the Ugly	NOT RATED	Western	161	[u'Clint Eastwood', u'Eli Wallach', u'Lee Van ...
26	8.6	Once Upon a Time in the West	PG-13	Western	175	[u'Henry Fonda', u'Charles Bronson', u'Claudia...
59	8.5	Django Unchained	R	Western	165	[u'Jamie Foxx', u'Christoph Waltz', u'Leonardo...
107	8.3	For a Few Dollars More	APPROVED	Western	132	[u'Clint Eastwood', u'Lee Van Cleef', u'Gian M...
119	8.3	Unforgiven	R	Western	131	[u'Clint Eastwood', u'Gene Hackman', u'Morgan ...
...	...	...	...	...	...	...
918	7.5	Running Scared	R	Action	122	[u'Paul Walker', u'Cameron Bright', u'Chazz Pa...
954	7.4	X-Men	PG-13	Action	104	[u'Patrick Stewart', u'Hugh Jackman', u'Ian Mc...
963	7.4	La Femme Nikita	R	Action	118	[u'Anne Parillaud', u'Marc Duret', u'Patrick F...
967	7.4	The Rock	R	Action	136	[u'Sean Connery', u'Nicolas Cage', u'Ed Harris']
976	7.4	Master and Commander: The Far Side of the World	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo...

979 rows × 6 columns

```
movies_df1.reset_index()
                        #reset_index will reset the index and one more column
                       #named "index" having previous index values will be added
                       # To delete this newly added "index" column, pass drop =True in paranthesis
```

	index	star_rating	title	content_rating	genre	duration	actors_list
0	954	7.4	X-Men	PG-13	Action	104	[u'Patrick Stewart', u'Hugh Jackman', u'Ian Mc...
1	963	7.4	La Femme Nikita	R	Action	118	[u'Anne Parillaud', u'Marc Duret', u'Patrick F...
2	967	7.4	The Rock	R	Action	136	[u'Sean Connery', u'Nicolas Cage', u'Ed Harris']
3	976	7.4	Master and Commander: The Far Side of the World	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo...
4	832	7.5	RoboCop	R	Action	102	[u'Peter Weller', u'Nancy Allen', u"Dan O'Herl...
...	...	...	...	...	...	...	...
974	107	8.3	For a Few Dollars More	APPROVED	Western	132	[u'Clint Eastwood', u'Lee Van Cleef', u'Gian M...
975	119	8.3	Unforgiven	R	Western	131	[u'Clint Eastwood', u'Gene Hackman', u'Morgan ...
976	59	8.5	Django Unchained	R	Western	165	[u'Jamie Foxx', u'Christoph Waltz', u'Leonardo...
977	26	8.6	Once Upon a Time in the West	PG-13	Western	175	[u'Henry Fonda', u'Charles Bronson', u'Claudia...
978	6	8.9	The Good, the Bad and the Ugly	NOT RATED	Western	161	[u'Clint Eastwood', u'Eli Wallach', u'Lee Van ...

979 rows × 7 columns

```
## To convert df to csv
movies_df1.to_csv("New_movie_data.csv")
```

```
movies_df1.set_index("title")
# We will get title column at the place of 0,1,2,3... column
```

	star_rating	content_rating	genre	duration	actors_list
title					
X-Men	7.4	PG-13	Action	104	[u'Patrick Stewart', u'Hugh Jackman', u'Ian Mc...
La Femme Nikita	7.4	R	Action	118	[u'Anne Parillaud', u'Marc Duret', u'Patrick F...
The Rock	7.4	R	Action	136	[u'Sean Connery', u'Nicolas Cage', u'Ed Harris']
Master and Commander: The Far Side of the World	7.4	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo...
RoboCop	7.5	R	Action	102	[u'Peter Weller', u'Nancy Allen', u"Dan O'Herl...
...	...	...	...	...	...
For a Few Dollars More	8.3	APPROVED	Western	132	[u'Clint Eastwood', u'Lee Van Cleef', u'Gian M...
Unforgiven	8.3	R	Western	131	[u'Clint Eastwood', u'Gene Hackman', u'Morgan ...
Django Unchained	8.5	R	Western	165	[u'Jamie Foxx', u'Christoph Waltz', u'Leonardo...
Once Upon a Time in the West	8.6	PG-13	Western	175	[u'Henry Fonda', u'Charles Bronson', u'Claudia...
The Good, the Bad and the Ugly	8.9	NOT RATED	Western	161	[u'Clint Eastwood', u'Eli Wallach', u'Lee Van ...

979 rows × 5 columns

```
emp_df.set_index("First Name")
# "First Name" column will be at the place index column
# we will have to drop unique valued column after analysis
```

First Name	Emp ID	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
Lois	677509	36.36	60.0	13.68	NaN	Denver
Brenda	940761	NaN	NaN	9.01	51063.0	Stonewall
Joe	428945	NaN	68.0	NaN	50155.0	Michigantown
Diane	438658	NaN	NaN	18.30	NaN	Hydetown
Benjamin	193819	40.31	58.0	4.01	117642.0	Fremont
...	...	...	...	...	...	...
Jose	639892	22.82	89.0	1.05	129774.0	Biloxi
Harold	704709	32.61	77.0	5.93	156194.0	Carol Stream
Nicole	461593	52.66	60.0	28.53	95673.0	Detroit
Theresa	392491	29.60	57.0	6.99	51015.0	Mc Grath
Tammy	495141	38.38	55.0	2.26	93650.0	Alma

100 rows × 6 columns

1. unique()

```
titanic_df["Survived"].unique()    # 0 & 1 will be unique values
```

```
array([0, 1], dtype=int64)
```

```
titanic_df["Pclass"].unique()   # output will be in array
```

```
array([3, 1, 2], dtype=int64)
```

```
## If values in target column are upto 5, then problem is of classification
## otherwise use regression
```

2 .nunique()

```
## to get the count of unique values
```

```
titanic_df["Survived"].nunique()
```

```
2
```

```
titanic_df["Pclass"].nunique()
```

```
3
```

```
titanic_df["PassengerId"].nunique()
```

```
891
```

3. value_counts()

```
titanic_df["Survived"].unique()
```

```
array([0, 1], dtype=int64)
```

```
titanic_df["Survived"].value_counts()
# by using value_counts() function, we will get the actual count of unique values in that target column
```

```
0    547
1    344
Name: Survived, dtype: int64
```

```
titanic_df["Pclass"].value_counts()
```

```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

```
titanic_df["Gender"].value_counts()
```

```
male      577
female    314
Name: Gender, dtype: int64
```

```
## To convert this data of unique value counts to dictionary, use to_dict()
```

```
titanic_df["Gender"].value_counts().to_dict()
```

```
{'male': 577, 'female': 314}
```

4. crosstab()

```
## To find the comparison between two columns
```

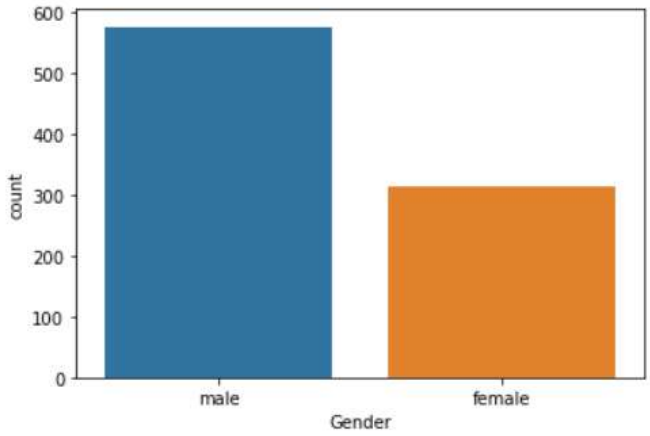
```
df1 = pd.crosstab(titanic_df["Gender"],titanic_df["Survived"])
df1
```

Survived	0	1
Gender		
female	81	233
male	466	111

```
import seaborn as sns
```

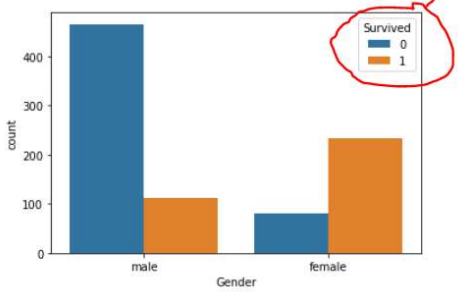
```
sns.countplot(x = titanic_df["Gender"])
```

```
<AxesSubplot:xlabel='Gender', ylabel='count'>
```



```
sns.countplot(x = titanic_df["Gender"],hue=titanic_df["Survived"])
```

```
<AxesSubplot:xlabel='Gender', ylabel='count'>
```



replace()

```
# To train the model, we need numeric values. This can be done by passing dictionary in replace function.
# So, these numeric values can be obtained by replacing the column values by particular number
# try to use replace function in series only.i.e for particular column instead of using this function for entire df
```

```
titanic_df.replace({"male" : 1,"female" : 0},inplace = False)
#replaced  for whole df
```

	PassengerId	Survived	Test	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	NaN	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	NaN	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	NaN	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	NaN	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	C123	S
4	5	0	NaN	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	NaN	S

```
titanic_df["Gender"].replace({"male" : 0, "female" : 1},inplace = False)
```

```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    1
889    0
890    0
Name: Gender, Length: 891, dtype: int64
```

```
titanic_df.replace({np.nan :1000},inplace = False)
## all the null values will be replaced by 1000
#no need to put inplace = False. inplace is False by default
```

	PassengerId	Survived	Test	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	1000.0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	1000	S
1	2	1	1000.0	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	1000.0	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	1000	S
3	4	1	1000.0	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	1000.0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	1000	S
...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	1000.0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	1000	S
887	888	1	1000.0	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	1000.0	3	Johnston, Miss. Catherine Helen "Carrie"	female	1000.0	1	2	W./C. 6607	23.4500	1000	S
889	890	1	1000.0	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C

```
titanic_df["Embarked"].value_counts()
```

```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

```
titanic_df["Embarked"].replace({"S":0,"C":1,"Q":2, np.nan : 0},inplace = True)
# here we have replaced null values by 0 just because there are 2 null values only and mode is S.ie.0
```

```
titanic_df
```

	PassengerId	Survived	Test	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	NaN	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	0.0
1	2	1	NaN	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	1.0
2	3	1	NaN	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	0.0
3	4	1	NaN	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	0.0
4	5	0	NaN	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	NaN	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	0.0
887	888	1	NaN	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	0.0
888	889	0	NaN	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	0.0
889	890	1	NaN	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	1.0
890	891	0	NaN	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	2.0

891 rows × 13 columns

astype

```
# To change the datatype from float to int or int to float
# we can't use object datatype here
```

```
titanic_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Test         0 non-null      float64
 3   Pclass       891 non-null    int64
 4   Name         891 non-null    object
 5   Gender       891 non-null    object
 6   Age          714 non-null    float64
 7   SibSp        891 non-null    int64
 8   Parch        891 non-null    int64
 9   Ticket       891 non-null    object
 10  Fare         891 non-null    float64
 11  Cabin        204 non-null    object
 12  Embarked     891 non-null    float64
dtypes: float64(4), int64(5), object(4)
memory usage: 90.6+ KB
```

```
titanic_df["Embarked"].astype(int)
```

```
0      0
1      1
2      0
3      0
4      0
      ..
886    0
887    0
888    0
889    1
890    2
Name: Embarked, Length: 891, dtype: int32
```

```
# Note : We can't change the datatype of a column even if a single null value is present in that column
```

```
titanic_df["Age"].fillna(titanic_df["Age"].mean(),inplace = True)
```

```
titanic_df["Age"].mean()
```

```
29.699117647058763
```

```
titanic_df["Age"].isna().sum()
```

```
0
```

```
# to convert datatype of age(float to int) use astype
titanic_df["Age"] = titanic_df["Age"].astype(int)
# Here we can't use inplace = True to update in original dataset.
# To do this change ,we need to assign it seperatly as did in this example
```

```
titanic_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Test         0 non-null      float64
 3   Pclass       891 non-null    int64
 4   Name         891 non-null    object
 5   Gender       891 non-null    object
 6   Age          891 non-null    int32
 7   SibSp        891 non-null    int64
 8   Parch        891 non-null    int64
 9   Ticket       891 non-null    object
 10  Fare         891 non-null    float64
 11  Cabin        204 non-null    object
 12  Embarked     891 non-null    float64
dtypes: float64(3), int32(1), int64(5), object(4)
memory usage: 87.1+ KB
```

```
## Now the data type of age column is changed to int
```

rename()

```
titanic_df
# By using rename fun, we can rename the column names(axis=1)
#or row names(axis =0 or no need to pass axis) by passing dictionary
```

	PassengerId	Survived	Test	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	NaN	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.2500	NaN	0.0
1	2	1	NaN	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	C85	1.0
2	3	1	NaN	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.9250	NaN	0.0
3	4	1	NaN	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1000	C123	0.0
4	5	0	NaN	3	Allen, Mr. William Henry	male	35	0	0	373450	8.0500	NaN	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	NaN	2	Montvila, Rev. Juozas	male	27	0	0	211536	13.0000	NaN	0.0
887	888	1	NaN	1	Graham, Miss. Margaret Edith	female	19	0	0	112053	30.0000	B42	0.0
888	889	0	NaN	3	Johnston, Miss. Catherine Helen "Carrie"	female	29	1	2	W./C. 6607	23.4500	NaN	0.0
889	890	1	NaN	1	Behr, Mr. Karl Howell	male	26	0	0	111369	30.0000	C148	1.0
890	891	0	NaN	3	Dooley, Mr. Patrick	male	32	0	0	370376	7.7500	NaN	2.0

891 rows × 13 columns

```
titanic_df.rename({"PassengerId": "ID","Name":"Passenger_name","Gender":"GENDER"},axis=1)
```

	ID	Survived	Test	Pclass	Passenger_name	GENDER	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	NaN	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.2500	NaN	0.0
1	2	1	NaN	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	C85	1.0

```
df = pd.DataFrame(np.random.randint(10,50,(6,4)),index=list("python"),columns = ["a","b","c","d"])
df
```

	a	b	c	d
p	11	31	20	38
y	31	17	49	24
t	17	15	30	41
h	43	40	47	11
o	24	26	45	41
n	41	14	12	25

```
df.rename({"p":"P","y":"Y"},inplace = True)
df
```

	a	b	c	d
P	11	31	20	38
Y	31	17	49	24
t	17	15	30	41
h	43	40	47	11
o	24	26	45	41
n	41	14	12	25

```
df.rename({"a":"A","b":"B"},axis = 1, inplace =True)
df
```

	A	B	c	d
P	11	31	20	38
Y	31	17	49	24

insert()

```
df.insert(index_number, column_name, value, allow_duplicate)
# By using insert function, we can add column at any required index position
# if column is added directly by defining a new column, then it will be added as the last column of df
```

```
df.insert(2,"Z",[10,20,30,40,50,60], allow_duplicates=True)
df
```

	A	B	Z	Z	c	d	E
P	11	31	10	10	20	38	1
Y	31	17	20	20	49	24	2
t	17	15	30	30	30	41	3
h	43	40	40	40	47	11	4
o	24	26	50	50	45	41	5
n	41	14	60	60	12	25	6

```
df.insert(2,"Z",[100,200,300,400,500,600],allow_duplicates=True)
df
```

	A	B	Z	Z	Z	c	d	E
P	11	31	100	10	10	20	38	1
Y	31	17	200	20	20	49	24	2
t	17	15	300	30	30	30	41	3
h	43	40	400	40	40	47	11	4
o	24	26	500	50	50	45	41	5
n	41	14	600	60	60	12	25	6

```
df = pd.DataFrame(np.random.randint(10,50,(6,4)),index=list("python"),columns = ["a","b","c","d"])
df
```

	a	b	c	d
p	17	43	10	24
y	45	31	31	43
t	49	11	34	20
h	29	39	46	13
o	27	20	25	46
n	43	48	37	29

```
# To replace a specific column, we can use rename function
# but for replacement of multiple columns at a time, assign a list of new column names to previous col names
# follow same for row names,i.e for index
```

```
df.columns = list('ABCD')
df.index = list('PYTHON')
df
```

	A	B	C	D
P	17	43	10	24
Y	45	31	31	43
T	49	11	34	20
H	29	39	46	13
O	27	20	25	46
N	43	48	37	29

```
df["E"] = np.arange(1,7)
df          # E column is added as the last column
```

	A	B	C	D	E
P	17	43	10	24	1
Y	45	31	31	43	2
T	49	11	34	20	3
H	29	39	46	13	4
O	27	20	25	46	5
N	43	48	37	29	6

```
## Adding of rows can be done by joining two different DataFrames
```

Apply Function

```
import pandas as pd
import numpy as np
```

```
df.apply(function_name)
```

```
# if we have to perform operations on column,such as(square of Age column is rquired)
#then it can be done by using lambda function in apply function.
#This can be done by using user defined function as well,but preferable is lambda function
```

```
array = np.random.randint(10,50,(6,6))
df =pd.DataFrame(array,columns = list("abcdef"))
df
```

	a	b	c	d	e	f
0	16	48	30	11	10	37
1	41	26	14	35	30	11
2	46	38	35	38	22	48
3	25	18	46	17	27	46
4	45	48	24	44	38	10
5	48	25	49	17	45	42

```
df.apply(lambda x : x**2)  #try to use lambda function instead of user defined function
```

	a	b	c	d	e	f
0	256	2304	900	121	100	1369
1	1681	676	196	1225	900	121
2	2116	1444	1225	1444	484	2304
3	625	324	2116	289	729	2116
4	2025	2304	576	1936	1444	100
5	2304	625	2401	289	2025	1764

```
def square(x):
    return x**2
df.apply(square)
```

	a	b	c	d	e	f
0	256	2304	900	121	100	1369
1	1681	676	196	1225	900	121
2	2116	1444	1225	1444	484	2304
3	625	324	2116	289	729	2116
4	2025	2304	576	1936	1444	100
5	2304	625	2401	289	2025	1764

```
df.apply(np.sqrt)
```

	a	b	c	d	e	f
0	4.000000	6.928203	5.477226	3.316625	3.162278	6.082763
1	6.403124	5.099020	3.741657	5.916080	5.477226	3.316625
2	6.782330	6.164414	5.916080	6.164414	4.690416	6.928203
3	5.000000	4.242641	6.782330	4.123106	5.196152	6.782330
4	6.708204	6.928203	4.898979	6.633250	6.164414	3.162278
5	6.928203	5.000000	7.000000	4.123106	6.708204	6.480741

```
#To apply a function for a specific column
```

```
df["a"] = df["a"].apply(np.sqrt)
df
```

	a	b	c	d	e	f
0	4.000000	48	30	11	10	37
1	6.403124	26	14	35	30	11
2	6.782330	38	35	38	22	48
3	5.000000	18	46	17	27	46
4	6.708204	48	24	44	38	10
5	6.928203	25	49	17	45	42

```
titanic_df["Gender"].apply(lambda x: x.upper())
```

```
0        MALE
1      FEMALE
2      FEMALE
3      FEMALE
4        MALE
        ...
886      MALE
887    FEMALE
888    FEMALE
889      MALE
890      MALE
Name: Gender, Length: 891, dtype: object
```

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....
5	8.9	12 Angry Men	NOT RATED	Drama	96	[u'Henry Fonda', u'Lee J. Cobb', u'Martin Bals...
6	8.9	The Good, the Bad and the Ugly	NOT RATED	Western	161	[u'Clint Eastwood', u'Eli Wallach', u'Lee Van ...
7	8.9	The Lord of the Rings: The Return of the King	PG-13	Adventure	201	[u'Elijah Wood', u'Viggo Mortensen', u'Ian McK...
8	8.9	Schindler's List	R	Biography	195	[u'Liam Neeson', u'Ralph Fiennes', u'Ben Kings...
9	8.9	Fight Club	R	Drama	139	[u'Brad Pitt', u'Edward Norton', u'Helena Bonh...
10	8.8	The Lord of the Rings: The Fellowship of the Ring	PG-13	Adventure	178	[u'Elijah Wood', u'Ian McKellen', u'Orlando Bl...
11	8.8	Inception	PG-13	Action	148	[u'Leonardo DiCaprio', u'Joseph Gordon-Levitt'...
12	8.8	Star Wars: Episode V - The Empire Strikes Back	PG	Action	124	[u'Mark Hamill', u'Harrison Ford', u'Carrie Fi...
13	8.8	Forrest Gump	PG-13	Drama	142	[u'Tom Hanks', u'Robin Wright', u'Gary Sinise']
14	8.8	The Lord of the Rings: The Two Towers	PG-13	Adventure	179	[u'Elijah Wood', u'Ian McKellen', u'Viggo Mort...

```
1 movies_df.groupby("genre").groups.keys()
```

```
dict_keys(['Action', 'Adventure', 'Animation', 'Biography', 'Comedy', 'Crime', 'Drama', 'Family', 'Fantasy', 'Film-Noir', 'His
ory', 'Horror', 'Mystery', 'Sci-Fi', 'Thriller', 'Western'])
```

```
1 movies_df.groupby("genre").groups
2 #it will form a group of genre and the index values of it will be given in a list.
```

```
{'Action': [3, 11, 12, 19, 20, 36, 37, 43, 44, 45, 75, 80, 82, 96, 100, 113, 118, 123, 129, 135, 138, 152, 163, 173, 177, 194,
196, 198, 235, 239, 240, 248, 261, 268, 276, 281, 296, 301, 312, 327, 349, 354, 366, 380, 385, 388, 391, 401, 403, 409, 411, 4
0, 433, 434, 437, 440, 452, 455, 469, 470, 488, 491, 502, 509, 515, 517, 529, 531, 532, 533, 534, 539, 552, 563, 567, 568, 570
```

```
1 movies_df.groupby("genre").first()
```

genre	star_rating	title	content_rating	duration	actors_list
Action	9.0	The Dark Knight	PG-13	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
Adventure	8.9	The Lord of the Rings: The Return of the King	PG-13	201	[u'Elijah Wood', u'Viggo Mortensen', u'Ian McK...
Animation	8.6	Spirited Away	PG	125	[u'Daveigh Chase', u'Suzanne Pleshette', u'Miy...
Biography	8.9	Schindler's List	R	195	[u'Liam Neeson', u'Ralph Fiennes', u'Ben Kings...
Comedy	8.6	Life Is Beautiful	PG-13	116	[u'Roberto Benigni', u'Nicoletta Braschi', u'G...
Crime	9.3	The Shawshank Redemption	R	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...

```
x_df = movies_df.groupby("genre").get_group("Action")
x_df.reset_index(drop=True)
#here we will get all the action movies
```

	star_rating	title	content_rating	genre	duration	actors_list
0	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
1	8.8	Inception	PG-13	Action	148	[u'Leonardo DiCaprio', u'Joseph Gordon-Levitt'...
2	8.8	Star Wars: Episode V - The Empire Strikes Back	PG	Action	124	[u'Mark Hamill', u'Harrison Ford', u'Carrie Fi...
3	8.7	Star Wars	PG	Action	121	[u'Mark Hamill', u'Harrison Ford', u'Carrie Fi...
4	8.7	The Matrix	R	Action	136	[u'Keanu Reeves', u'Laurence Fishburne', u'Car...
...	...	...	...	...	...	...
131	7.5	Running Scared	R	Action	122	[u'Paul Walker', u'Cameron Bright', u'Chazz Pa...
132	7.4	X-Men	PG-13	Action	104	[u'Patrick Stewart', u'Hugh Jackman', u'Ian Mc...
133	7.4	La Femme Nikita	R	Action	118	[u'Anne Parillaud', u'Marc Duret', u'Patrick F...
134	7.4	The Rock	R	Action	136	[u'Sean Connery', u'Nicolas Cage', u'Ed Harris']
135	7.4	Master and Commander: The Far Side of the World	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo...

136 rows × 6 columns

```
y_df = movies_df.groupby("genre").get_group("Biography")
y_df.reset_index(drop=True)
```

	star_rating	title	content_rating	genre	duration	actors_list
0	8.9	Schindler's List	R	Biography	195	[u'Liam Neeson', u'Ralph Fiennes', u'Ben Kings...
1	8.7	Goodfellas	R	Biography	146	[u'Robert De Niro', u'Ray Liotta', u'Joe Pesci']
2	8.6	The Intouchables	R	Biography	112	[u'Fran\xe7ois Cluzet', u'Omar Sy', u'Anne Le ...

```
movies_df.groupby("content_rating").get_group("APPROVED").reset_index(drop=True).head(5)
# column name should be in groupby()
```

	star_rating	title	content_rating	genre	duration	actors_list
0	8.7	It's a Wonderful Life	APPROVED	Drama	130	[u'James Stewart', u'Donna Reed', u'Lionel Bar...
1	8.6	Rear Window	APPROVED	Mystery	112	[u'James Stewart', u'Grace Kelly', u'Wendell C...
2	8.5	The Great Dictator	APPROVED	Comedy	125	[u'Charles Chaplin', u'Paulette Goddard', u'Ja...
3	8.5	Paths of Glory	APPROVED	Drama	88	[u'Kirk Douglas', u'Ralph Meeker', u'Adolphe M...
4	8.4	Witness for the Prosecution	APPROVED	Crime	116	[u'Tyrone Power', u'Marlene Dietrich', u'Charl...

```
movies_df.groupby(["genre","content_rating"]).get_group(("Action","R")).reset_index(drop=True).head(5)
#for passing multiple column names in groupby(),use list to pass column names
#for passing multiple names in a particular column in get_group(),use tuple to pass multiple names.
#these names in get_group should be passed in the same sequence as that of columns are passed in groupby fun.
```

	star_rating	title	content_rating	genre	duration	actors_list
0	8.7	The Matrix	R	Action	136	[u'Keanu Reeves', u'Laurence Fishburne', u'Car...
1	8.6	Saving Private Ryan	R	Action	169	[u'Tom Hanks', u'Matt Damon', u'Tom Sizemore']
2	8.5	Gladiator	R	Action	155	[u'Russell Crowe', u'Joaquin Phoenix', u'Conni...
3	8.5	Terminator 2: Judgment Day	R	Action	137	[u'Arnold Schwarzenegger', u'Linda Hamilton', ...
4	8.4	Aliens	R	Action	137	[u'Sigourney Weaver', u'Michael Biehn', u'Carr...

```
# to compare two columns , we use crosstab function
pd.crosstab(movies_df["genre"],movies_df["content_rating"])
```

content_rating	APPROVED	G	GP	NC-17	NOT RATED	PASSED	PG	PG-13	R	TV-MA	UNRATED	X
genre												
Action	3	1	1	0	4	1	11	44	67	0	3	0
Adventure	3	2	0	0	5	1	21	23	17	0	2	0
Animation	3	20	0	0	3	0	25	5	5	0	1	0
Biography	1	2	1	0	1	0	6	29	36	0	0	0

```
# as a data scientist we ourselves need to decide categorical and continuous data
```

```
movies_df.groupby(["genre","content_rating"]).get_group(("Action","R")).reset_index(drop=True).head(5).sort_values("star_
```

	star_rating	title	content_rating	genre	duration	actors_list
4	8.4	Aliens	R	Action	137	[u'Sigourney Weaver', u'Michael Biehn', u'Carr...
2	8.5	Gladiator	R	Action	155	[u'Russell Crowe', u'Joaquin Phoenix', u'Conni...
3	8.5	Terminator 2: Judgment Day	R	Action	137	[u'Arnold Schwarzenegger', u'Linda Hamilton', ...
1	8.6	Saving Private Ryan	R	Action	169	[u'Tom Hanks', u'Matt Damon', u'Tom Sizemore']
0	8.7	The Matrix	R	Action	136	[u'Keanu Reeves', u'Laurence Fishburne', u'Car...

```
movies_df.groupby(['genre','content_rating']).get_group(('Drama','PG-13'))[['genre','title','duration']].head(10)
```

	genre	title	duration
13	Drama	Forrest Gump	142
53	Drama	The Prestige	130
73	Drama	Jodaeiye Nader az Simin	123
146	Drama	Warrior	140
180	Drama	Hotel Rwanda	121
191	Drama	The Sixth Sense	107
211	Drama	Departures	130
227	Drama	Million Dollar Baby	132
232	Drama	Chungking Express	98
259	Drama	The Help	146

```
1 movies_df.groupby(['genre'])['duration',"star_rating"].max().head(15)
```

```
C:\Users\Amol\AppData\Local\Temp/ipykernel_13944/937022236.py:1: FutureWarni
ed to a tuple of keys) will be deprecated, use a list instead.
  movies_df.groupby(['genre'])['duration',"star_rating"].max().head(15)
```

genre	duration	star_rating
Action	205	9.0
Adventure	224	8.9
Animation	134	8.6
Biography	202	8.9
Comedy	187	8.6
Crime	229	9.3
Drama	242	8.9
Family	115	7.9
Fantasy	112	7.7
Film-Noir	111	8.3
History	66	8.0
Horror	146	8.6
Mystery	160	8.6
Sci-Fi	132	8.2
Thriller	120	8.0

```
movies_df.loc[movies_df['genre'] == "Adventure"]
```

	star_rating	title	content_rating	genre	duration	actors_list
7	8.9	The Lord of the Rings: The Return of the King	PG-13	Adventure	201	[u'Elijah Wood', u'Viggo Mortensen', u'Ian McK...
10	8.8	The Lord of the Rings: The Fellowship of the Ring	PG-13	Adventure	178	[u'Elijah Wood', u'Ian McKellen', u'Orlando Bl...
14	8.8	The Lord of the Rings: The Two Towers	PG-13	Adventure	179	[u'Elijah Wood', u'Ian McKellen', u'Viggo Mort...
15	8.7	Interstellar	PG-13	Adventure	169	[u'Matthew McConaughey', u'Anne Hathaway', u'J...
54	8.5	Back to the Future	PG	Adventure	116	[u'Michael J. Fox', u'Christopher Lloyd', u'Le...
...	...	...	...	...	...	...
936	7.4	True Grit	NaN	Adventure	128	[u'John Wayne', u'Kim Darby', u'Glen Campbell']
937	7.4	Labyrinth	PG	Adventure	101	[u'David Bowie', u'Jennifer Connelly', u'Toby ...
943	7.4	The Bucket List	PG-13	Adventure	97	[u'Jack Nicholson', u'Morgan Freeman', u'Sean ...
953	7.4	The NeverEnding Story	PG	Adventure	102	[u'Noah Hathaway', u'Barret Oliver', u'Tami St...
975	7.4	Back to the Future Part III	PG	Adventure	118	[u'Michael J. Fox', u'Christopher Lloyd', u'Ma...

75 rows × 6 columns

```
# We will get same result by loc function and groupby function
```

```
movies_df.groupby('genre').get_group('Adventure')
```

	star_rating	title	content_rating	genre	duration	actors_list
7	8.9	The Lord of the Rings: The Return of the King	PG-13	Adventure	201	[u'Elijah Wood', u'Viggo Mortensen', u'Ian McK...
10	8.8	The Lord of the Rings: The Fellowship of the Ring	PG-13	Adventure	178	[u'Elijah Wood', u'Ian McKellen', u'Orlando Bl...
14	8.8	The Lord of the Rings: The Two Towers	PG-13	Adventure	179	[u'Elijah Wood', u'Ian McKellen', u'Viggo Mort...
15	8.7	Interstellar	PG-13	Adventure	169	[u'Matthew McConaughey', u'Anne Hathaway', u'J...
54	8.5	Back to the Future	PG	Adventure	116	[u'Michael J. Fox', u'Christopher Lloyd', u'Le...

```
movies_df.loc[(movies_df['genre'] == "Adventure")|(movies_df['genre'] == "Action")]
```

	star_rating	title	content_rating	genre	duration	actors_lis
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E.
7	8.9	The Lord of the Rings: The Return of the King	PG-13	Adventure	201	[u'Elijah Wood', u'Viggo Mortensen', u'Ian McK.
10	8.8	The Lord of the Rings: The Fellowship of the Ring	PG-13	Adventure	178	[u'Elijah Wood', u'Ian McKellen', u'Orlando Bl.
11	8.8	Inception	PG-13	Action	148	[u'Leonardo DiCaprio', u'Joseph Gordon-Levitt'.
12	8.8	Star Wars: Episode V - The Empire Strikes Back	PG	Action	124	[u'Mark Hamill', u'Harrison Ford', u'Carrie Fi.
...	...	...	...	...	...	.
954	7.4	X-Men	PG-13	Action	104	[u'Patrick Stewart', u'Hugh Jackman', u'Ian Mc.
963	7.4	La Femme Nikita	R	Action	118	[u'Anne Parillaud', u'Marc Duret', u'Patrick F.
967	7.4	The Rock	R	Action	136	[u'Sean Connery', u'Nicolas Cage', u'Ed Harris
975	7.4	Back to the Future Part III	PG	Adventure	118	[u'Michael J. Fox', u'Christopher Lloyd', u'Ma.
976	7.4	Master and Commander: The Far Side of the World	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo.

211 rows × 6 columns

```
movies_df.loc[(movies_df['genre'] == "Adventure") & (movies_df['content_rating'] == "PG")]
# two different conditions must be passed in round brackets
```

	star_rating	title	content_rating	genre	duration	actors_list
54	8.5	Back to the Future	PG	Adventure	116	[u'Michael J. Fox', u'Christopher Lloyd', u'Le...
85	8.4	Lawrence of Arabia	PG	Adventure	216	[u"Peter O'Toole", u'Alec Guinness', u'Anthony...
101	8.3	Monty Python and the Holy Grail	PG	Adventure	91	[u'Graham Chapman', u'John Cleese', u'Eric Idle']
137	8.3	The Bridge on the River Kwai	PG	Adventure	161	[u'William Holden', u'Alec Guinness', u'Jack H...
142	8.3	Lagaan: Once Upon a Time in India	PG	Adventure	224	[u'Aamir Khan', u'Gracy Singh', u'Rachel Shell...

```
movies_df.loc[movies_df["duration"]<=180]
#in loc function, use condition for that particular column
```

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....
5	8.9	12 Angry Men	NOT RATED	Drama	96	[u'Henry Fonda', u'Lee J. Cobb', u'Martin Bals...
...	...	...	...	...	...	...
974	7.4	Tootsie	PG	Comedy	116	[u'Dustin Hoffman', u'Jessica Lange', u'Teri G...
975	7.4	Back to the Future Part III	PG	Adventure	118	[u'Michael J. Fox', u'Christopher Lloyd', u'Ma...
976	7.4	Master and Commander: The Far Side of the World	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo...
977	7.4	Poltergeist	PG	Horror	114	[u'JoBeth Williams', u"Heather O'Rourke", u'Cr...
978	7.4	Wall Street	R	Crime	126	[u'Charlie Sheen', u'Michael Douglas', u'Tamar...

947 rows × 6 columns

```
movies_df.loc[(movies_df["duration"]>=120) & (movies_df["duration"]<=180) ]
# two different conditions must be passed in round brackets
```

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....
6	8.9	The Good, the Bad and the Ugly	NOT RATED	Western	161	[u'Clint Eastwood', u'Eli Wallach', u'Lee Van ...

****

****

```
x_df = movies_df.loc[(movies_df['duration'] >= 120) & (movies_df['duration'] <= 180)
      & (movies_df['star_rating'] > 8.5)
      & (movies_df['content_rating'] == 'R')
    & (movies_df['genre'] == 'Crime')]

x_df

#here we have applied 4 different conditions
#this is the advantage of loc function
```

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....
21	8.7	City of God	R	Crime	130	[u'Alexandre Rodrigues', u'Matheus Nachtergael...

```
x_df = movies_df.loc[(movies_df['duration'] >= 120) & (movies_df['duration'] <= 180)
      & (movies_df['star_rating'] > 8.5)
      & (movies_df['content_rating'] == 'R'),['title','actors_list']].reset_index(drop = True)

x_df
```

	title	actors_list
0	The Shawshank Redemption	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	The Godfather	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	Pulp Fiction	[u'John Travolta', u'Uma Thurman', u'Samuel L....
3	Fight Club	[u'Brad Pitt', u'Edward Norton', u'Helena Bonh...
4	One Flew Over the Cuckoo's Nest	[u'Jack Nicholson', u'Louise Fletcher', u'Mich...
5	Goodfellas	[u'Robert De Niro', u'Ray Liotta', u'Joe Pesci']

```
col_list = ['star_rating','title','content_rating','genre']
#By using 'usecols = col_list' while reading main file and by creating col_list seperatly,
#we can directly read the required columns only while reading the file
movies_df = pd.read_csv(file_path,usecols= col_list)
movies_df
```

	star_rating	title	content_rating	genre
0	9.3	The Shawshank Redemption	R	Crime
1	9.2	The Godfather	R	Crime
2	9.1	The Godfather: Part II	R	Crime
3	9.0	The Dark Knight	PG-13	Action
4	8.9	Pulp Fiction	R	Crime
...	...	...	...	...
974	7.4	Tootsie	PG	Comedy
975	7.4	Back to the Future Part III	PG	Adventure
976	7.4	Master and Commander: The Far Side of the World	PG-13	Action
977	7.4	Poltergeist	PG	Horror
978	7.4	Wall Street	R	Crime

979 rows × 4 columns

****

****

```
df = pd.DataFrame(np.random.randint(30,50,size = (6,3)))
df.to_csv('test.csv',header = False,index = False)
```

```
df1 = pd.read_csv('test.csv')
df1
#here we are getting column names as values in rows(34,46,36),which is not expected
#To avoid this, pass attribute "header = None"
```

	34	46	36
0	49	41	45
1	39	32	32
2	46	36	41
3	35	38	42
4	48	32	30

```
df1 = pd.read_csv('test.csv',header=None)
df1
```

	0	1	2
0	34	46	36
1	49	41	45
2	39	32	32
3	46	36	41
4	35	38	42
5	48	32	30

```
movies_df = pd.read_csv(file_path,index_col = "title")
movies_df
#we can directly set the required index column while reading the file by using index_col = "title"
```

title	star_rating	content_rating	genre	duration	actors_list
The Shawshank Redemption	9.3	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
The Godfather	9.2	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
The Godfather: Part II	9.1	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
The Dark Knight	9.0	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
Pulp Fiction	8.9	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....
...	...	...	...	...	...
Tootsie	7.4	PG	Comedy	116	[u'Dustin Hoffman', u'Jessica Lange', u'Teri G...
Back to the Future Part III	7.4	PG	Adventure	118	[u'Michael J. Fox', u'Christopher Lloyd', u'Ma...
Master and Commander: The Far Side of the World	7.4	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo...
Poltergeist	7.4	PG	Horror	114	[u'JoBeth Williams', u"Heather O'Rourke", u'Cr...
Wall Street	7.4	R	Crime	126	[u'Charlie Sheen', u'Michael Douglas', u'Tamar...

979 rows × 5 columns

delimiter

```
df = pd.DataFrame(np.random.randint(10,50,(5,4)))
df.to_csv("Sample")
```

```
Sample_df = pd.read_csv('Sample')
Sample_df
```

	|0|1|2|3
0	0|23|10|29|43
1	1|13|19|28|37
2	2|30|48|15|35
3	3|26|13|28|17
4	4|38|49|31|15

```
Sample_df = pd.read_csv("Sample",delimiter = "|" , header = None )
Sample_df
```

	0	1	2	3	4
0	NaN	0	1	2	3
1	0.0	23	10	29	43
2	1.0	13	19	28	37
3	2.0	30	48	15	35
4	3.0	26	13	28	17
5	4.0	38	49	31	15

```
df = pd.read_excel(file_path)
df
#There are 3 different sheets in this excel file.By default we can read here the first sheet
#To read the second or other sheet that you want to read,just pass the attribute "sheet_name = "
```

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509	Lois	36.36	60.0	13.68	NaN	Denver
1	940761	Brenda	NaN	NaN	9.01	51063.0	Stonewall
2	428945	Joe	NaN	68.0	NaN	50155.0	Michigantown
3	408351	Diane	NaN	NaN	18.30	NaN	Hydetown
4	193819	Benjamin	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...
95	639892	Jose	22.82	89.0	1.05	129774.0	Biloxi
96	704709	Harold	32.61	77.0	5.93	156194.0	Carol Stream
97	461593	Nicole	52.66	60.0	28.53	95673.0	Detroit
98	392491	Theresa	29.60	57.0	6.99	51015.0	Mc Grath
99	495141	Tammy	38.38	55.0	2.26	93650.0	Alma

100 rows × 7 columns

```
df = pd.read_excel(file_path, sheet_name= "Sheet1")
df
```

	Employee Names	Salary
0	sdhg	20000
1	sghsgj	50000
2	jsjfsg	40000
3	sfj	30000

```
df1 = movies_df.groupby("genre").get_group("Crime")
df1
#Here you will get all the movies whose genre is "Crime"
```

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....
21	8.7	City of God	R	Crime	130	[u'Alexandre Rodrigues', u'Matheus Nachtergael...
...	...	...	...	...	...	...
927	7.5	Brick	R	Crime	110	[u'Joseph Gordon-Levitt', u'Lukas Haas', u'Emi...
931	7.4	Mean Streets	R	Crime	112	[u'Robert De Niro', u'Harvey Keitel', u'David ...
950	7.4	Bound	R	Crime	108	[u'Jennifer Tilly', u'Gina Gershon', u'Joe Pan...
969	7.4	Law Abiding Citizen	R	Crime	109	[u'Gerard Butler', u'Jamie Foxx', u'Leslie Bibb']
978	7.4	Wall Street	R	Crime	126	[u'Charlie Sheen', u'Michael Douglas', u'Tamar...

124 rows × 6 columns

```
df1.to_csv("Movies_details")
```

```
df = pd.read_excel(file_path, sheet_name= "Sheet1")
df
```

	Employee Names	Salary
0	sdhg	20000
1	sghsgj	50000
2	jsjfsg	40000
3	sfj	30000
4	hsfgsh	38000

```
df.to_csv("Numeric Data.csv",index = False)
# if we don't want index column, put index = False
# if we don't want columns names , use header = False,it will automatically put 0,1,2,3... as column names
```

```
# to create and write nultiple sheets in excel file,use ExcelWriter function
#and add each sheet one by one and then at last save as "writer.save()"
```

```
writer = pd.ExcelWriter("UPDATED_EMP_DATA1.xlsx",engine = "xlsxwriter")
df1.to_excel(writer,sheet_name="Names")
df2.to_excel(writer,sheet_name="Age in cmp")
df3.to_excel(writer,sheet_name="salary")
writer.save()
```

```
# if index = false, index column will be deleted
```

```
data = df.to_dict(orient = "list")
data
# we can convert dataframe to dictionary
```

```
{'a': [26, 97, 44, 82, 12, 57, 21],
 'b': [24, 32, 44, 65, 36, 83, 36],
 'c': [43, 18, 91, 60, 19, 84, 29],
 'd': [50, 92, 13, 46, 29, 42, 93]}
```

```
data = df.to_dict(orient = "records")
data
# "orient = records" will return list of dictionaries(Useful in MongoDB)
```

```
[{'a': 26, 'b': 24, 'c': 43, 'd': 50},
 {'a': 97, 'b': 32, 'c': 18, 'd': 92},
 {'a': 44, 'b': 44, 'c': 91, 'd': 13},
 {'a': 82, 'b': 65, 'c': 60, 'd': 46},
 {'a': 12, 'b': 36, 'c': 19, 'd': 29},
 {'a': 57, 'b': 83, 'c': 84, 'd': 42},
 {'a': 21, 'b': 36, 'c': 29, 'd': 93}]
```

JSON

```
JSON file is nothing but a structure of dictionary
```

```
df.to_json('numbers.json')  # front end developers
```

```
df.to_json('numbers1.json',orient = 'records')
```

Pandas Joins

```
1. append:
       1.1 df1.append(df2)
       1.2 df1.append([df_list],ignore_index=False)
       1.3 df1.append(series)

2. concat:
    2.1. pd.concat(df_list,axis = 0,join = 'outer',ignore_index=False)
    2.1. pd.concat(df_list,axis = 1,join = 'outer',ignore_index=False)

3. merge:
    pd.merge(df1,df2,how = 'inner',on = 'ColName')
    df1.merge(df2,how = 'inner',on = 'ColName')

4. join:
    df1.join(df2,how = 'left')

Joins >> {'inner','outer','left','right'}
```

1. append()

```
1 df1.append(df2)
```

df1

	Name	Age	Location
0	Akash	20	Pune
1	Swapnil	30	Mumbai
2	Pravin	40	Delhi
3	Rohit	25	Chennai
4	Virat	36	Bangalore

df2

	Name	Age	Location
0	Suraj	20	Hyderabad
1	Ajay	30	Mumbai
2	Rohit	25	Chennai
3	Vijay	40	Delhi
4	Sagar	36	Kolkata

```
1 df2.append(df1)
```

df2

	Name	Age	Location
0	Suraj	20	Hyderabad
1	Ajay	30	Mumbai
2	Rohit	25	Chennai
3	Vijay	40	Delhi
4	Sagar	36	Kolkata

df1

	Name	Age	Location
0	Akash	20	Pune
1	Swapnil	30	Mumbai
2	Pravin	40	Delhi
3	Rohit	25	Chennai
4	Virat	36	Bangalore

```
df2.append(df1).reset_index(drop=True)
# To reset index use "reset_index" function and pass drop = True
# result of "reset_index" function and "ignore_index=True" function is same..
# just the difference is we need to pass ignore_index fun after the df which is to be appended..
```

	Name	Age	Location	Designation
0	Suraj	20	Hyderabad	Python Developer
1	Ajay	30	Mumbai	Data Scientist
2	Rohit	25	Chennai	JAVA Devloper
3	Vijay	40	Delhi	Data Analyst
4	Sagar	36	Kolkata	ML Engineer
5	Akash	20	Pune	NaN
6	Swapnil	30	Mumbai	NaN
7	Pravin	40	Delhi	NaN
8	Rohit	25	Chennai	NaN
9	Virat	36	Bangalore	NaN

```
df2.append(df1,ignore_index=True)
```

	Name	Age	Location	Designation
0	Suraj	20	Hyderabad	Python Developer
1	Ajay	30	Mumbai	Data Scientist
2	Rohit	25	Chennai	JAVA Devloper
3	Vijay	40	Delhi	Data Analyst
4	Sagar	36	Kolkata	ML Engineer
5	Akash	20	Pune	NaN
6	Swapnil	30	Mumbai	NaN
7	Pravin	40	Delhi	NaN
8	Rohit	25	Chennai	NaN
9	Virat	36	Bangalore	NaN

join multiple dataframes using append function

```
df1 = pd.DataFrame({"Name" : ['Akash','Swapnil','Pravin','Rohit','Virat'],
                    'Age':[20,30,40,25,36],
                    'Location':['Pune','Mumbai','Delhi','Chennai','Bangalore']})

df2 = pd.DataFrame({"Name" : ['Suraj','Ajay','Rohit','Vijay','Sagar'],
                    'Age':[20,30,25,40,36],
                    'Location':['Hyderabad','Mumbai','Chennai','Delhi','Kolkata'],
                    'Designation':['Python Developer','Data Scientist','JAVA Devloper',
                                   'Data Analyst', 'ML Engineer']})
df3 = pd.DataFrame({"Name" : ['Suraj','Ajay','Rohit','Vijay','Sagar'],
                    'Age':[20,30,25,40,36],
                    'Location':['Hyderabad','Mumbai','Chennai','Delhi','Kolkata'],
                    'Salary':[50000,60000,70000,40000,55000]})

df1.append([df2,df3],ignore_index=True)

#if there are multiple dataframes then we need to pass those dfs in a list which are to be appended
```

	Name	Age	Location	Designation	Salary
0	Akash	20	Pune	NaN	NaN
1	Swapnil	30	Mumbai	NaN	NaN
2	Pravin	40	Delhi	NaN	NaN
3	Rohit	25	Chennai	NaN	NaN
4	Virat	36	Bangalore	NaN	NaN
5	Suraj	20	Hyderabad	Python Developer	NaN
6	Ajay	30	Mumbai	Data Scientist	NaN
7	Rohit	25	Chennai	JAVA Devloper	NaN
8	Vijay	40	Delhi	Data Analyst	NaN
9	Sagar	36	Kolkata	ML Engineer	NaN
10	Suraj	20	Hyderabad	NaN	50000.0
11	Ajay	30	Mumbai	NaN	60000.0
12	Rohit	25	Chennai	NaN	70000.0

add single row in dataframe

```
df1 = pd.DataFrame({"Name" : ['Akash','Swapnil','Pravin','Rohit','Virat'],
                    'Age':[20,30,40,25,36],
                    'Location':['Pune','Mumbai','Delhi','Chennai','Bangalore']})

df2 = pd.DataFrame({"Name" : ['Ajay'],
                    'Age':[32],
                    'Location':['Kolkata']})
df1.append(df2,ignore_index=True)
# The row which is to be added using df must have the values in a list/array/tuple (sequential data)
# for that dictionary even if a single value is to be passed..
```

	Name	Age	Location
0	Akash	20	Pune
1	Swapnil	30	Mumbai
2	Pravin	40	Delhi
3	Rohit	25	Chennai
4	Virat	36	Bangalore
5	Ajay	32	Kolkata

create series

```
s1 = pd.Series({"Name" : 'Ajay',
                'Age':32,
                'Location':'Kolkata'})
s1
```

```
Name            Ajay
Age               32
Location     Kolkata
dtype: object
```

```
s1 = pd.Series([2,3,4,5,6])
s1
# we can pass tuple also...
```

```
0    2
1    3
2    4
3    5
4    6
dtype: int64
```

```
df3
```

	Name	Age	Location	Salary
0	Suraj	20	Hyderabad	50000
1	Ajay	30	Mumbai	60000
2	Rohit	25	Chennai	70000
3	Vijay	40	Delhi	40000
4	Sagar	36	Kolkata	55000

```
s3 = pd.Series({"Name" : 'Ajay',
                'Age':32,
                'Location':'Kolkata'})
s3
```

```
Name            Ajay
Age               32
Location     Kolkata
dtype: object
```

```
# in joining df,if u may have to add single row or column, then in such case, use Series..
```

```
df3.append(s3,ignore_index=True)
```

	Name	Age	Location	Salary
0	Suraj	20	Hyderabad	50000.0
1	Ajay	30	Mumbai	60000.0
2	Rohit	25	Chennai	70000.0
3	Vijay	40	Delhi	40000.0
4	Sagar	36	Kolkata	55000.0
5	Ajay	32	Kolkata	NaN

```
df3
```

	Name	Age	Location	Salary
0	Suraj	20	Hyderabad	50000
1	Ajay	30	Mumbai	60000
2	Rohit	25	Chennai	70000
3	Vijay	40	Delhi	40000
4	Sagar	36	Kolkata	55000

```
df3['Designation'] = ['Python Developer','Data Scientist','JAVA Devloper',
                      'Data Analyst', 'ML Engineer']
df3
# To add new column, df[Col_name] = [column data in list]..
# here the length of the list must be equal to the number of rows...
```

	Name	Age	Location	Salary	Designation
0	Suraj	20	Hyderabad	50000	Python Developer
1	Ajay	30	Mumbai	60000	Data Scientist
2	Rohit	25	Chennai	70000	JAVA Devloper
3	Vijay	40	Delhi	40000	Data Analyst
4	Sagar	36	Kolkata	55000	ML Engineer

```
#if the length of the list is more than number of rows, which is to be added
# in the column which is to be added in the df,then use series
df1 = pd.DataFrame({"Name" : ['Akash','Swapnil','Pravin','Rohit','Virat'],
                    'Age':[20,30,40,25,36],
                    'Location':['Pune','Mumbai','Delhi','Chennai','Bangalore']})

df1['Designation'] =  pd.Series(['Python Developer','Data Scientist','JAVA Devloper',
                                 'Data Analyst','ML',"DL"])
df1
# here len(list) = 6 > number of rows(5)
# still there is no error because we used series as a column which is to be added in df
# if len(list) < no of rows, remaining values in the column will be NaN , if we used series
```

	Name	Age	Location	Designation
0	Akash	20	Pune	Python Developer
1	Swapnil	30	Mumbai	Data Scientist
2	Pravin	40	Delhi	JAVA Devloper
3	Rohit	25	Chennai	Data Analyst
4	Virat	36	Bangalore	ML

```
new_df = pd.concat([df1,df2],axis = 0,ignore_index=True)
new_df
# by default axis = 0, df2 will be added below df1(in rows)
#df1.append(df2) will give same result as pd.concat([df1,df2])
```

	Name	Age	Location
0	Akash	20	Pune
1	Swapnil	30	Mumbai
2	Pravin	40	Delhi
3	Rohit	25	Chennai
4	Virat	36	Bangalore
5	Suraj	20	Hyderabad
6	Ajay	30	Mumbai
7	Rohit	25	Chennai
8	Vijay	40	Delhi
9	Sagar	36	Kolkata

```
new_df = pd.concat([df1,df2],axis = 1)
new_df
# if axis = 1, new df will be added by using columns
# index is very imp in concat fun
```

	Name	Age	Location	Name	Age	Location
0	Akash	20	Pune	Suraj	20	Hyderabad
1	Swapnil	30	Mumbai	Ajay	30	Mumbai
2	Pravin	40	Delhi	Rohit	25	Chennai
3	Rohit	25	Chennai	Vijay	40	Delhi
4	Virat	36	Bangalore	Sagar	36	Kolkata

```
df1 = pd.DataFrame({"Name" : ['Akash','Swapnil','Pravin','Rohit','Virat'],
                    'Age':[20,30,40,25,36],
                    'Location':['Pune','Mumbai','Delhi','Chennai','Bangalore']})

df2 = pd.DataFrame({"Name" : ['Suraj','Ajay','Rohit','Vijay','Sagar'],
                    'Age':[20,30,25,40,36],
                    'Location':['Hyderabad','Mumbai','Chennai','Delhi','Kolkata'],
                    'Designation':['Python Developer','Data Scientist','JAVA Devloper',
                                   'Data Analyst', 'ML Engineer']})
df3 = pd.DataFrame({"Name" : ['Suraj','Ajay','Rohit','Vijay','Sagar'],
                    'Age':[20,30,25,40,36],
                    'Location':['Hyderabad','Mumbai','Chennai','Delhi','Kolkata'],
                    'Salary':[50000,60000,70000,40000,55000]})

new_df = pd.concat([df1,df2,df3],axis = 0,ignore_index = True)
new_df
```

	Name	Age	Location	Designation	Salary
0	Akash	20	Pune	NaN	NaN
1	Swapnil	30	Mumbai	NaN	NaN
2	Pravin	40	Delhi	NaN	NaN
3	Rohit	25	Chennai	NaN	NaN
4	Virat	36	Bangalore	NaN	NaN
5	Suraj	20	Hyderabad	Python Developer	NaN
6	Ajay	30	Mumbai	Data Scientist	NaN
7	Rohit	25	Chennai	JAVA Devloper	NaN
8	Vijay	40	Delhi	Data Analyst	NaN
9	Sagar	36	Kolkata	ML Engineer	NaN
10	Suraj	20	Hyderabad	NaN	50000.0
11	Ajay	30	Mumbai	NaN	60000.0
12	Rohit	25	Chennai	NaN	70000.0
13	Vijay	40	Delhi	NaN	40000.0
14	Sagar	36	Kolkata	NaN	55000.0

```
df1 = pd.DataFrame({"Name" : ['Akash','Swapnil','Pravin','Rohit','Virat'],
                    'Age':[20,30,40,25,36],
                    'Location':['Pune','Mumbai','Delhi','Chennai','Bangalore']}, index = list("ABCDE"))

df2 = pd.DataFrame({"Name" : ['Suraj','Ajay','Rohit','Vijay'],
                    'Age':[20,30,25,40],
                    'Location':['Hyderabad','Mumbai','Chennai','Delhi'],
                    'Designation':['Python Developer','Data Scientist',
                                   'Data Analyst', 'ML Engineer']}, index =list("xyzA"))

new_df = pd.concat([df1,df2],axis = 1,ignore_index = False)
new_df
# for axis = 1,if index name is different for dataframes to be concated,
# then NaN values will be there in those columns for respective index names
```

	Name	Age	Location	Name	Age	Location	Designation
A	Akash	20.0	Pune	Vijay	40.0	Delhi	ML Engineer
B	Swapnil	30.0	Mumbai	NaN	NaN	NaN	NaN
C	Pravin	40.0	Delhi	NaN	NaN	NaN	NaN
D	Rohit	25.0	Chennai	NaN	NaN	NaN	NaN
E	Virat	36.0	Bangalore	NaN	NaN	NaN	NaN
x	NaN	NaN	NaN	Suraj	20.0	Hyderabad	Python Developer
y	NaN	NaN	NaN	Ajay	30.0	Mumbai	Data Scientist
z	NaN	NaN	NaN	Rohit	25.0	Chennai	Data Analyst

```
new_df.iloc[3]
# To acces particular row by loc function
```

```
Name              Rohit
Age                25.0
Location        Chennai
Name                NaN
Age                 NaN
Location            NaN
Designation         NaN
Name: D, dtype: object
```

```
new_df[5:6]
# To access 5th row by slicing
```

	Name	Age	Location	Name	Age	Location	Designation
x	NaN	NaN	NaN	Suraj	20.0	Hyderabad	Python Developer

```
new_df.loc[['y',"D"]]
# To access particular rows we can use iloc function,loc function and slicing method
```

	Name	Age	Location	Name	Age	Location	Designation
y	NaN	NaN	NaN	Ajay	30.0	Mumbai	Data Scientist
D	Rohit	25.0	Chennai	NaN	NaN	NaN	NaN

```
# for concat, join = outer is default >> it will give same result as that of regular join/concat
# join = inner >> intersection of dfs >> null value rows will be deleted(axis =1)
# join = inner >> intersection of dfs >> only common columns will be there in new df if axis = 0 is used
```

```
df1 = pd.DataFrame({"Name" : ['Akash','Swapnil','Pravin','Rohit','Virat'],
                    'Age':[20,30,40,25,36],
                    'Location':['Pune','Mumbai','Delhi','Chennai','Bangalore']})

df2 = pd.DataFrame({"Name" : ['Suraj','Ajay','Rohit','Vijay'],
                    'Age':[20,30,25,40],
                    'Location':['Hyderabad','Mumbai','Chennai','Delhi'],
                    'Designation':['Python Developer','Data Scientist',
                                   'Data Analyst', 'ML Engineer']})

new_df = pd.concat([df1,df2],axis = 0,join = "inner",ignore_index = False)
new_df
# here the column (Designation) having null values(after concatination) in it is not added
```

	Name	Age	Location
0	Akash	20	Pune
1	Swapnil	30	Mumbai
2	Pravin	40	Delhi
3	Rohit	25	Chennai
4	Virat	36	Bangalore

```
df1 = pd.DataFrame({"Name" : ['Akash','Swapnil','Pravin','Rohit','Virat'],
                    'Age':[20,30,40,25,36],
                    'Location':['Pune','Mumbai','Delhi','Chennai','Bangalore']})

df2 = pd.DataFrame({"Name" : ['Suraj','Ajay','Rohit','Vijay'],
                    'Age':[20,30,25,40],
                    'Location':['Hyderabad','Mumbai','Chennai','Delhi'],
                    'Designation':['Python Developer','Data Scientist',
                                   'Data Analyst', 'ML Engineer']})

new_df = pd.concat([df1,df2],axis = 1,join = "inner",ignore_index = False)
new_df
# Here the row having null values is deleted
```

	Name	Age	Location	Name	Age	Location	Designation
0	Akash	20	Pune	Suraj	20	Hyderabad	Python Developer
1	Swapnil	30	Mumbai	Ajay	30	Mumbai	Data Scientist
2	Pravin	40	Delhi	Rohit	25	Chennai	Data Analyst
3	Rohit	25	Chennai	Vijay	40	Delhi	ML Engineer

```
df1 = pd.DataFrame({"Name" : ['Akash','Swapnil','Pravin','Rohit','Virat'],
                    'Age':[20,30,40,25,36],
                    'Location':['Pune','Mumbai','Delhi','Chennai','Bangalore']})

df2 = pd.DataFrame({"Name" : ['Suraj','Ajay','Rohit','Vijay'],
                    'Age':[20,30,25,40],
                    'Location':['Hyderabad','Mumbai','Chennai','Delhi'],
                    'Designation':['Python Developer','Data Scientist',
                                   'Data Analyst', 'ML Engineer']})

new_df = pd.concat([df1,df2],axis = 1,join = "outer",ignore_index = False)
new_df
# join = outer does not deal with intersection concept.
# for join = outer >> in output, there will be all the rows/columns irrespective of null values
```

	Name	Age	Location	Name	Age	Location	Designation
0	Akash	20	Pune	Suraj	20.0	Hyderabad	Python Developer
1	Swapnil	30	Mumbai	Ajay	30.0	Mumbai	Data Scientist
2	Pravin	40	Delhi	Rohit	25.0	Chennai	Data Analyst
3	Rohit	25	Chennai	Vijay	40.0	Delhi	ML Engineer
4	Virat	36	Bangalore	NaN	NaN	NaN	NaN

```
df1 = pd.DataFrame({"Name" : ['Akash','Swapnil','Pravin','Rohit','Virat'],
                    'Age':[20,30,40,25,36],
                    'Location':['Pune','Mumbai','Delhi','Chennai','Bangalore']},
                    index = list('abcde'))

df2 = pd.DataFrame({"Name" : ['Suraj','Ajay','Rohit','Vijay'],
                    'Age':[20,30,25,40],
                    'Location':['Hyderabad','Mumbai','Chennai','Delhi'],
                    'Designation':['Python Developer','Data Scientist',
                                   'Data Analyst', 'ML Engineer']}, index = list('xyda'))

new_df = pd.concat([df1,df2],axis = 1,join= 'outer')
new_df
# if row labels are same then it will be printed only once
```

	Name	Age	Location	Name	Age	Location	Designation
a	Akash	20.0	Pune	Vijay	40.0	Delhi	ML Engineer
b	Swapnil	30.0	Mumbai	NaN	NaN	NaN	NaN
c	Pravin	40.0	Delhi	NaN	NaN	NaN	NaN
d	Rohit	25.0	Chennai	Rohit	25.0	Chennai	Data Analyst
e	Virat	36.0	Bangalore	NaN	NaN	NaN	NaN
x	NaN	NaN	NaN	Suraj	20.0	Hyderabad	Python Developer
y	NaN	NaN	NaN	Ajay	30.0	Mumbai	Data Scientist

```
df1 = pd.DataFrame({"Name" : ['Akash','Swapnil','Pravin','Rohit','Virat'],
                    'Age':[20,30,40,25,36],
                    'Location':['Pune','Mumbai','Delhi','Chennai','Bangalore']},
                    index = list('abcde'))

df2 = pd.DataFrame({"Name" : ['Suraj','Ajay','Rohit','Vijay'],
                    'Age':[20,30,25,40],
                    'Location':['Hyderabad','Mumbai','Chennai','Delhi'],
                    'Designation':['Python Developer','Data Scientist',
                                   'Data Analyst', 'ML Engineer']}, index = list('xyad'))

new_df = pd.concat([df1,df2],axis = 1,join= 'inner')
new_df
# Here only common rows are in output because axis = 1 and join = inner
```

	Name	Age	Location	Name	Age	Location	Designation
a	Akash	20	Pune	Rohit	25	Chennai	Data Analyst
d	Rohit	25	Chennai	Vijay	40	Delhi	ML Engineer

```
df3
```

	Name	Age	Location	Salary
0	Suraj	20	Hyderabad	50000
1	Ajay	30	Mumbai	60000
2	Rohit	25	Chennai	70000
3	Vijay	40	Delhi	40000
4	Sagar	36	Kolkata	55000

```
pd.concat([df1,df2,df3],axis = 1,join = 'inner')
# Here we won't get a single row in output because there is not any common row label from df1,df2,df3
```

Name	Age	Location	Name	Age	Location	Designation	Name	Age	Location	Salary

```
df1 = pd.DataFrame({"Name" : ['Akash','Swapnil','Pravin','Rohit','Virat'],
                    'Age':[20,30,40,25,36],
                    'Location':['Pune','Mumbai','Delhi','Chennai','Bangalore']},
                    index = list('abcde'))

df2 = pd.DataFrame({"Name" : ['Suraj','Ajay','Rohit','Vijay'],
                    'Age':[20,30,25,40],
                    'Location':['Hyderabad','Mumbai','Chennai','Delhi'],
                    'Designation':['Python Developer','Data Scientist',
                                   'Data Analyst', 'ML Engineer']}, index = list('xyda'))

df3 = pd.DataFrame({"Name" : ['Suraj','Ajay','Rohit','Vijay','Sagar'],
                    'Age':[20,30,25,40,36],
                    'Location':['Hyderabad','Mumbai','Chennai','Delhi','Kolkata'],
                    'Salary':[50000,60000,70000,40000,55000]},list('abmno'))

pd.concat([df1,df2,df3],axis = 1,join = 'inner')
# only 'a' row is common in all three dfs (df1,df2,df3), hence it will be in output as we used join = "inner"
```

	Name	Age	Location	Name	Age	Location	Designation	Name	Age	Location	Salary
a	Akash	20	Pune	Vijay	40	Delhi	ML Engineer	Suraj	20	Hyderabad	50000

```
# For conversion of categorical data to numeric data(ex. male>>1,female>>0), we use one hot encoding method
# for this we use get.dummies function for that particular column
```

****

****

```
# For conversion of categorical data to numeric data(ex. male>>1,female>>0), we use one hot encoding method
# for this we use get.dummies function for that particular column
```

```
file_path = r"E:\Python_Learning\Class Notes\JUNE\06_10_Pandas_apply\titanic.csv"
file_path
```

```
'E:\\Python_Learning\\Class Notes\\JUNE\\06_10_Pandas_apply\\titanic.csv'
```

```
titanic_df = pd.read_csv(file_path)
titanic_df.head()
```

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
Gender_df = pd.get_dummies(titanic_df["Gender"])
Gender_df.head()
```

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1

3. Merge

```
left_df.merge(right_df)

pd.merge(left_df, right_df,how = 'inner')

# how = "inner" is default for merge function
# Drawback of merge function is that, we can use only two dfs.
# Whereas in case of append and concat we can use multiple dfs
```

```
df1 = pd.DataFrame({"Name" : ['Akash','Swapnil','Pravin','Rohit','Virat'],
                    'Age':[20,30,40,25,36],
                    'Location':['Pune','Mumbai','Delhi','Chennai','Bangalore']},)

df2 = pd.DataFrame({"Name" : ['Suraj','Swapnil','Rohit','Vijay'],
                    'Age':[20,30,25,40],
                    'Location':['Hyderabad','Mumbai','Chennai','Delhi'],
                    'Designation':['Python Developer','Data Scientist',
                                   'Data Analyst', 'ML Engineer']})
df2
```

	Name	Age	Location	Designation
0	Suraj	20	Hyderabad	Python Developer
1	Swapnil	30	Mumbai	Data Scientist
2	Rohit	25	Chennai	Data Analyst
3	Vijay	40	Delhi	ML Engineer

```
df1.merge(df2)
# how = "inner" is default in merge fun
```

	Name	Age	Location	Designation
0	Swapnil	30	Mumbai	Data Scientist
1	Rohit	25	Chennai	Data Analyst

```
pd.merge(df1,df2)
# we can merge in above syntax also, result will be same
```

	Name	Age	Location	Designation
0	Swapnil	30	Mumbai	Data Scientist
1	Rohit	25	Chennai	Data Analyst

```
pd.merge(df1,df2,how = 'outer')
# here common rows will be once in output
# reset_index is automatic here
```

	Name	Age	Location	Designation
0	Akash	20	Pune	NaN
1	Swapnil	30	Mumbai	Data Scientist
2	Pravin	40	Delhi	NaN
3	Rohit	25	Chennai	Data Analyst
4	Virat	36	Bangalore	NaN
5	Suraj	20	Hyderabad	Python Developer
6	Vijay	40	Delhi	ML Engineer

```
pd.merge(df2,df1,how = 'left')
```

	Name	Age	Location	Designation
0	Suraj	20	Hyderabad	Python Developer
1	Swapnil	30	Mumbai	Data Scientist
2	Rohit	25	Chennai	Data Analyst
3	Vijay	40	Delhi	ML Engineer

```
pd.merge(df1,df2,how = 'left')
```

	Name	Age	Location	Designation
0	Akash	20	Pune	NaN
1	Swapnil	30	Mumbai	Data Scientist
2	Pravin	40	Delhi	NaN
3	Rohit	25	Chennai	Data Analyst
4	Virat	36	Bangalore	NaN

```
pd.merge(df1,df2,how = 'left',on = 'Name')
# on = "Name" >> this column will be common column in dfs
# Age and Location column is present in both dataframes
# for df1 >> Age_x,Location_x  > we can change these suffixes too..
# for df2 >> Age_y,Location_y
# Designation column was in only one df
```

	Name	Age_x	Location_x	Age_y	Location_y	Designation
0	Akash	20	Pune	NaN	NaN	NaN
1	Swapnil	30	Mumbai	30.0	Mumbai	Data Scientist
2	Pravin	40	Delhi	NaN	NaN	NaN
3	Rohit	25	Chennai	25.0	Chennai	Data Analyst
4	Virat	36	Bangalore	NaN	NaN	NaN

```
pd.merge(df1,df2,how = 'left',on = 'Name',suffixes=("_df1","_df2"))
```

	Name	Age_df1	Location_df1	Age_df2	Location_df2	Designation
0	Akash	20	Pune	NaN	NaN	NaN
1	Swapnil	30	Mumbai	30.0	Mumbai	Data Scientist
2	Pravin	40	Delhi	NaN	NaN	NaN
3	Rohit	25	Chennai	25.0	Chennai	Data Analyst
4	Virat	36	Bangalore	NaN	NaN	NaN

```
# Drawback of merge function is that, we can use only two dfs.
# Whereas in case of append and concat we can use multiple dfs.
```

```
df1 = pd.DataFrame({'X':[20,30,40],'Y':[100,200,300]},index = list('abc'))
df2 = pd.DataFrame({'A':[2000,30000,4000],'B':[100000,20000,30000]},index = list('acd'))

df2
```

	A	B
a	2000	100000
c	30000	20000
d	4000	30000

```
df1
```

	X	Y
a	20	100
b	30	200
c	40	300

```
df1.join(df2,how = 'left')
# default >> How = left
# all rows and values of df1
```

	X	Y	A	B
a	20	100	2000.0	100000.0
b	30	200	NaN	NaN
c	40	300	30000.0	20000.0

```
df1.join(df2,how = 'left')
# default >> How = left
# all rows and values of df1
```

	X	Y	A	B
a	20	100	2000.0	100000.0
b	30	200	NaN	NaN
c	40	300	30000.0	20000.0

```
df1.join(df2,how = 'right')
# all rows and values of df2
```

	X	Y	A	B
a	20.0	100.0	2000	100000
c	40.0	300.0	30000	20000
d	NaN	NaN	4000	30000

```
df1.join(df2,how = 'outer')
# will get whole dataset
```

	X	Y	A	B
a	20.0	100.0	2000.0	100000.0
b	30.0	200.0	NaN	NaN
c	40.0	300.0	30000.0	20000.0
d	NaN	NaN	4000.0	30000.0

```
df1.join(df2,how = 'inner')
# will get common rows only
```

	X	Y	A	B
a	20	100	2000	100000
c	40	300	30000	20000

```
df = pd.DataFrame(np.random.randint(30,50,size = (6,4)))
df.columns = list('abcd')
df.index = list('python')

df
# We can set row and column labels by assigning seperately as mentioned above or
# also this labelling can be at the time of forming a df
```

	a	b	c	d
p	34	48	49	37
y	34	30	46	49
t	43	49	33	39
h	35	30	47	47
o	32	39	47	43
n	39	31	47	44

```
df = pd.DataFrame(np.random.randint(30,50,size = (6,4)),columns = list('ABCD'),index = list('PYTHON'))
df
```

	A	B	C	D
P	36	31	39	42
Y	46	46	30	33
T	45	47	32	30
H	37	49	41	38
O	33	43	43	47
N	36	41	41	37

items()

```
for col_name,column_data in df.items():
    print(col_name)
```

```
A
B
C
D
```

```
df1 = pd.DataFrame(np.random.randint(30,50,size = (6,4)))
df1
for index,column in df1.items():
    print(index)
```

```
0
1
2
3
```

```
for col_name,column_data in df.items():
    print(column_data)
    print("*"*60)
# column data will be in output in a series
```

```
P    36
Y    46
T    45
H    37
O    33
N    36
Name: A, dtype: int32
************************************************************
P    31
Y    46
T    47
```

iteritems()

```
for col_name,column_data in df.iteritems():
    print(column_data)
    print("*"*60)
# items() and iteritems() are having same functionality
```

```
P    36
Y    46
T    45
H    37
O    33
N    36
Name: A, dtype: int32
************************************************************
P    31
Y    46
T    47
H    49
O    43
N    41
Name: B, dtype: int32
************************************************************
P    39
Y    30
T    32
H    41
O    43
N    41
Name: C, dtype: int32
************************************************************
P    42
Y    33
T    30
H    38
```

```
df
```

	A	B	C	D
P	36	31	39	42
Y	46	46	30	33
T	45	47	32	30
H	37	49	41	38
O	33	43	43	47
N	36	41	41	37

```
for index,row in df.iterrows():
    print(index)
```

```
P
Y
T
H
O
N
```

```
for index,row in df.iterrows():
    print(row)
    print("*"*70)

# row data will be in output in a series
```

```
A    36
B    31
C    39
D    42
Name: P, dtype: int32
**********************************************************************
A    46
B    46
C    30
D    33
Name: Y, dtype: int32
**********************************************************************
```

1. to_numpy()

```
1 array1 = np.random.randint(20,30,size = (5,3))
2 array1
```

```
array([[24, 29, 27],
       [22, 23, 24],
       [23, 21, 27],
       [27, 23, 28],
       [23, 21, 25]])
```

```
1 df = pd.DataFrame(array1)
2 df
```

	0	1	2
0	24	29	27
1	22	23	24
2	23	21	27
3	27	23	28
4	23	21	25

```
1 df.to_numpy()
```

```
array([[24, 29, 27],
       [22, 23, 24],
       [23, 21, 27],
       [27, 23, 28],
       [23, 21, 25]])
```

array to list

```
1 array1.tolist()
2 # we will get nested list
```

```
[[24, 29, 27], [22, 23, 24], [23, 21, 27], [27, 23, 28], [23, 21, 25]]
```

```
df.to_numpy().tolist()

# df >> numpy >> list
```

```
[[24, 29, 27], [22, 23, 24], [23, 21, 27], [27, 23, 28], [23, 21, 25]]
```

2. Values

```
emp_df = pd.read_csv(file_path)
emp_df.head(7)
```

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509	Lois	36.36	60.0	13.68	NaN	Denver
1	940761	Brenda	NaN	NaN	9.01	51063.0	Stonewall
2	428945	Joe	NaN	68.0	NaN	50155.0	Michigantown
3	408351	Diane	NaN	NaN	18.30	NaN	Hydetown
4	193819	Benjamin	NaN	NaN	NaN	NaN	NaN
5	499687	Patrick	34.86	58.0	12.02	98189.0	Macksburg
6	193819	NaN	NaN	50.0	0.87	98189.0	Atlanta

```
emp_df.head().to_numpy()
```

```
array([[677509, 'Lois', 36.36, 60.0, 13.68, nan, 'Denver'],
       [940761, 'Brenda', nan, nan, 9.01, 51063.0, 'Stonewall'],
       [428945, 'Joe', nan, 68.0, nan, 50155.0, 'Michigantown'],
       [408351, 'Diane', nan, nan, 18.3, nan, 'Hydetown'],
       [193819, 'Benjamin', nan, nan, nan, nan, nan]], dtype=object)
```

```
emp_df.head().values
#to_numpy and values is same
# values function does not require paranthesis after it
```

```
array([[677509, 'Lois', 36.36, 60.0, 13.68, nan, 'Denver'],
       [940761, 'Brenda', nan, nan, 9.01, 51063.0, 'Stonewall'],
       [428945, 'Joe', nan, 68.0, nan, 50155.0, 'Michigantown'],
       [408351, 'Diane', nan, nan, 18.3, nan, 'Hydetown'],
```

Drop duplicates

```
df
```

	0	1	2
0	24	29	27
1	22	23	24
2	23	21	27
3	27	23	28
4	23	21	25

```
df.iloc[2,0] = 24
df.iloc[2,1] = 29
df.iloc[2,2] = 27
df
```

	0	1	2
0	24	29	27
1	22	23	24
2	24	29	27
3	27	23	28
4	23	21	25

```
df.drop_duplicates()
# duplicate rows will be dropped
```

	0	1	2
0	24	29	27
1	22	23	24
3	27	23	28
4	23	21	25

Access column and rows from dataframe

```
:
1  df[col_name]  # series
2  df[col_list]  # df
3
4  df[2:3]  # 2nd rows
5  df[5:10]  # 5 to 9 rows
6
7  df.iloc[row_index, col_index]
8  df.iloc[row_index]
9  df.iloc[3:5, 2:4] # 3rd and 4th row + 2nd and 3rd col
10 df.iloc[[2,5,6], [5,6,7,1]]
11
12
13 df.loc[row_label, col_label]
14 df.loc[row_label]
15 df.loc[2:5,5:8]  # 2,3,4,5 rows and 5,6,7,8 columns
16 df.loc[1:5,5]    # series
17
18 df.head()   # to access first n rows, default value is 5
19 df.head(7)  # first 7 rows
20
21 df.tail()   # to access last n rows, default value is 5
22 df.tail(10)
23
24 df.columns   # all column names
25 df.index     # all rows
26 df.axes      # rows and columns
```

Information about dataframe

```
1. df.shape
2. df.dtypes
3. df.info()
4. df.describe() # stats
```

Handling of Missing values

```
1. Detect Missing values:
    1. df.isna()
    2. df.isnull()
    3. df.isna().sum()
    4. df.isnull().sum()
    5. df.isna().mean()
    6. df.isna().mean() * 100

2. Fill Missing Values:
    1. df.fillna()
    2. Fill Missing values by using stats

3. Drop Column and rows using missing values threshold:
    1. df.dropna(axis = 0,thresh = df.shape[1])
    2. df.dropna(axis = 1,thresh = 100)
```

Statistics

```
1. df[col_name].mean()
2. df[col_name].median()
3. df[col_name].mode()[0]
4. df[col_name].std()
5. df[col_name].var()

6. df[col_name].max()
7. df[col_name].min()

8. df.mean()
9. df.median()
10. df.mode()
11. df.mean()
12. df.std()
13. df.var()
```

Sorting

```
1. df.sort_index()
2. df.sort_values()
3. df.reset_index()
4. df.set_index()
```

Other Functions

```
1. df[col_name].unique()
2. df[col_name].nunique()
3. df[col_name].value_counts()
4. pd.crosstab(col1,col2)
5. replace:
    1. df.replace(dict)
    2. df[col_name].replace(dict)

6. rename:
    1. df.rename(dict_col_labels, axis=1,inplace = True)
    2. df.rename(dict_row_labels, axis=0,inplace = True)

7. df.insert(index, values, allow_duplicate = True)

8. df.drop_duplicates()  # used to drop duplicate rows from df

9. df[col_name].astype()

10. df.apply()

11. groupby:
    1. df.groupby().groups
    2. df.groupby().first()
    3. df.groupby().get_group()

12. pd.getdummies()

13. pivot_table
14. map
```

Read and Write Data

```
Read Data:
    1. pd.read_csv(index_col, header,)
    2. pd.read_excel(index_col,sheet_name, header)
    3. pd.read_json()
    4. pd.ExcelWriter()

Create Files using df:
    1. df.to_csv(csv_file_path,index = True, header = True)
    2. df.to_excel(excel_file_path, index = True, header = True)
    3. df.to_json(orient = records)
    4. df.to_dict(orient = {records,list,index})
```

Join DataFrames

```
1. append:
    1. df1.append(df2)
    2. df1.append(df_list)

2. concat:
    1. pd.concat(df_list,join = 'outer',axis = 0) # join = [inner , outer]

3. merge:
    1. pd.merge(left_df,right_df,how = 'inner')
    2. left_df.merge(right_df,how = 'inner')
    how = ['inner','outer','left','right']

4. join:
    1. left_df.join(right_df,how = 'left')
    how = ['inner','outer','left','right']

```